Θεσσαλονικη 2009

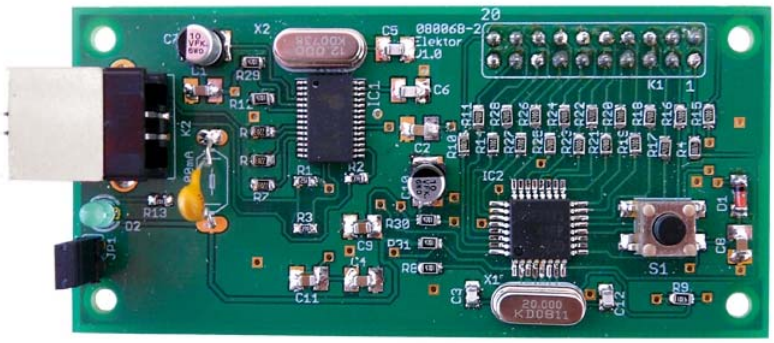
ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

Πτυχιακη Εργασια Σπουδαστων :

ΓΚΟΥΤΡΑ ΑΘΑΝΑΣΙΟΥ
ΣΟΥΛΗ ΠΑΝΑΓΙΩΤΗ

Θεμα :

“ΕΛΕΓΧΟΣ ΚΙΝΗΤΗΡΑ ΚΑΙ ΜΕΤΡΗΣΕΙΣ ΘΕΡΜΟΚΡΑΣΙΑΣ ΜΕΣΩ ΔΙΑΔΙΚΤΥΟΥ”



Επιβλεπων Καθηγητης : **Αλεξανδρής Αλέξανδρος**

Επιβλεπων καθηγητης
Αλεξανδρης Αλεξανδρος, Ηλεκτρονικός, Ανωτέρα Σχολή Ηλεκτρονικών.
Μεταπτυχιακό στη Διασφάλιση Ποιότητας (ΕΑΠ)

Σπουδαστες
Γκουτρας Αθανασιος Κ.Α.Σ 599604
Σουλης Παναγιωτης Κ.Α.Σ 599336

Αναληψη και περατωση πτυχιακης εργασιας 2007 - 2009

Περίληψη

Η εργασία αυτή περιελάμβανε την υλοποίηση ενός micro Web Server του ***MSC1210 – CS 8900A*** ο οποίος έχει τη δυνατότητα ελέγχου μιας σειράς αναλογικών αισθητήρων και συσκευών και να επικοινωνεί μέσω κάρτας δικτύου στο διαδικτυο. Το υλικό μέρος του συστήματος αυτού είναι ελάχιστο, μόνο δύο ολοκληρωμένα χρειάζονται για ολόκληρο τον web server , η πλακέτα μετρήσεων ακριβείας βασισμένη στον MSC 1210 και η πλακέτα επέκτασης γύρω από το ολοκληρωμένο οδηγό δικτύου Ethernet CS8900A Το ενδιαφέρον επικεντρώνεται στην κατασκευή της πλακέτας A/D την κατασκευή της πλακέτας δικτύου την ανάπτυξη λογισμικού για τον προγραμματισμό των δυο πλακετών (A/D & πλακέτα δικτύου). Επίσης έχουμε κατασκευή Html ιστοσελίδα, όπου εκεί θα γίνεται και ο έλεγχος της όλης κατασκευής, υλοποίηση πλακέτας τροφοδοτικού.

Η εργασία αυτή έχει δυνατότητα επέκτασης με άλλες 7 αναλογικές εισόδους με πολλές εφαρμογές όπως μετεωρολογική χρήση, έλεγχος συσκευών θερμοκρασίας, χρήση σε συστήματα θέρμανσης και κλιματισμού και θερμοκήπια.

Summary

This work concluded the build of a micro web server the MSC1210 – CS8900A who has the ability to control a number of analog devises and sensors and also communicate through a web card with the World Wide Web (internet).
The hardware of this devise is the minimum, only two electronic circuits needed for the whole server. The measurement circuit is based on the net driver Ethernet CS8900A. Our Interest is focused on the build of the analog / digital circuit, the Ethernet card also the development of the software which will program the two circuits (A/D & Ethernet card). In the end we made the Html code which will allow us to control the whole web server.

This web server has also another 7 analog inputs which will allow us to expand its use, with a lot of applications, like:
Meteorological use, control of temperature and air condition devises

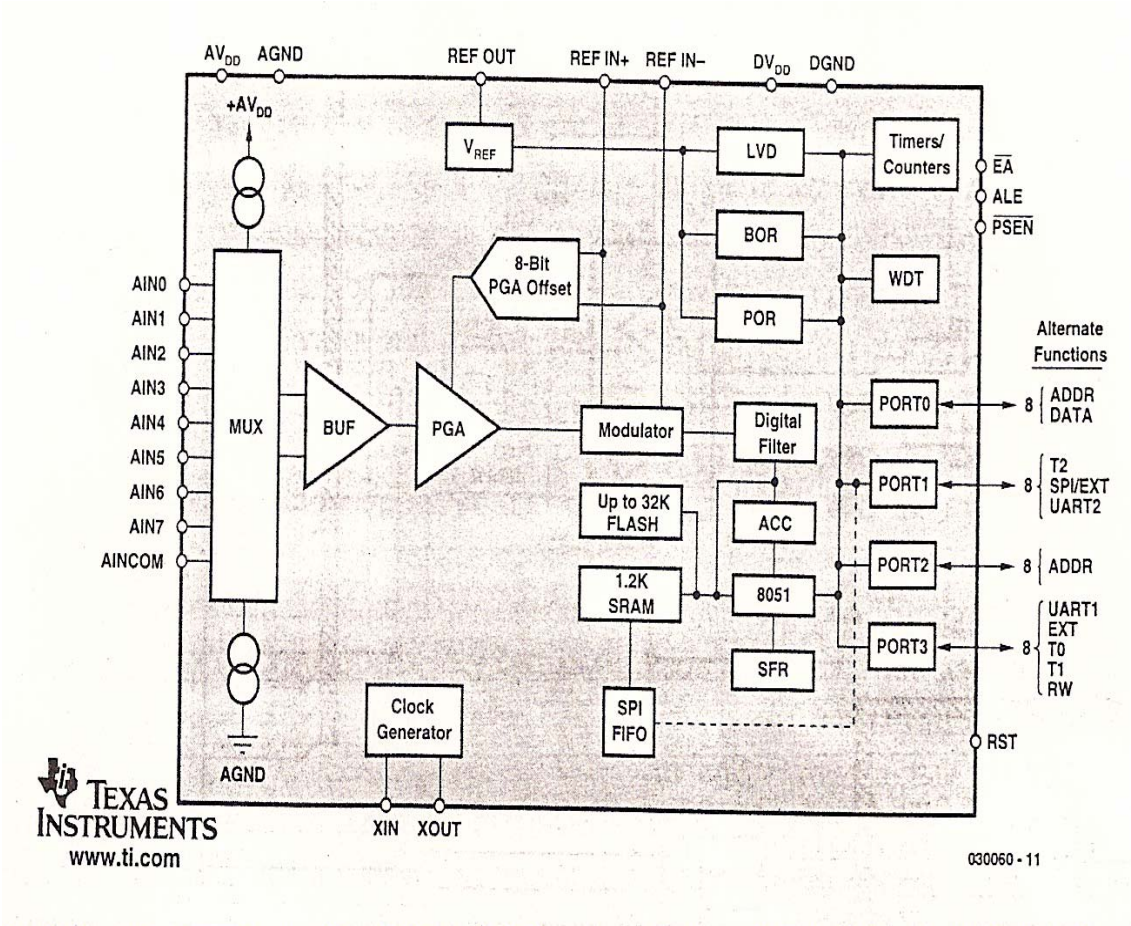
Πρόλογος

Χρησιμοποιώντας ως μέσο το διαδίκτυο είναι εφικτή η τηλεμετρία και ο απομακρυσμένος έλεγχος. Το σύστημα που παρουσιάζουμε είναι ευέλικτο, γρήγορο, μικρό σε μέγεθος, κατασκευάζεται εύκολα και παρέχει υψηλές επιδόσεις. Αποτελεί έναν αρκετά ευέλικτο κόμβο στο διαδίκτυο, παρά τις μικρές διαστάσεις του και την μέτρια πολυπλοκότητά του. Αποτελείται από μια κάρτα μικροελεγκτή με μια μονάδα προσαρμογής δικτύου. Χάρη στη συμπαγή κατασκευή του αλλά και στην ευελιξία της κάρτας του μικροελεγκτή, ο Micro Web Server αποτελεί την ιδανική λύση για εφαρμογές έλεγχου και μετρήσεων από οποιοδήποτε σημείο του κόσμου μέσω του διαδικτύου..

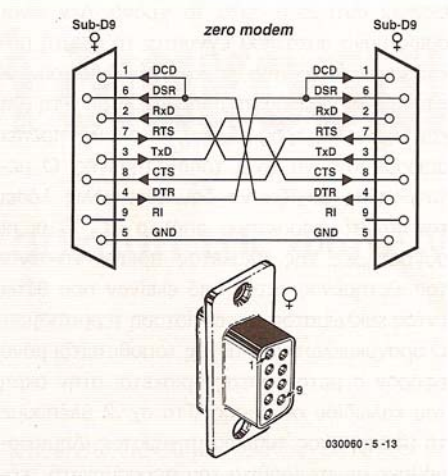
.

1. Πλακέτα MSC 1210 της Texas Instruments:

Η καρδιά της πλακέτας μας είναι ο μικροελεγκτής MSC 1210 της Texas Instruments ο οποίος επιτρέπει ακριβείς μετρήσεις τάσεων από οκτώ διαφορετικές πηγές. Η σημερινή αγορά των μικροελεγκτών βρίθει από παράγωγα – κλώνους του 8051. Τα περισσότερα από αυτά έχουν να επιδείξουν πολλές περισσότερες δυνατότητες από ότι το αρχικό προϊόν της Intel. Κάποια μάλιστα είναι εξοπλισμένα με μετατροπείς αναλογικού σήματος σε ψηφιακό ικανούς να γεφυρώσουν ψηφιακά δεδομένα με τον αναλογικό κόσμο. Ο MSC 1210 είναι ένας μικροελεγκτής που εκτός από τον κορυφαίων επιδόσεων ενσωματωμένο μετατροπέα A / D, έχει να επιδείξει και πολλά ακόμα προσόντα. Το μικρό μέγεθος της δικαιολογεί με τη σειρά του τις εξ ίσου μικρές διαστάσεις των εξαρτημάτων (τα περισσότερα είναι SMD) που την αποτελούν. Για την υψηλή ακρίβεια των μετρήσεων δεν ευθύνεται μόνο το υλικό μέρος της κατασκευής αλλά φυσικά και το λογισμικό που γράφτηκε για την συγκεκριμένη εφαρμογή πάνω στη γλώσσα C. Οι εσωτερικές μονάδες του MSC 1210 φαίνονται παραστατικά στο πιο κάτω σχήμα



Η πιο σημαντική απ' όλες είναι αυτή του μετατροπέα αναλογικού σε ψηφιακό των 24 ψηφίων / 8 εισόδων. Ακολουθεί η βαθμίδα παραγωγής τάσης αναφοράς υψηλής θερμοκρασιακής σταθερότητας και φυσικά ο πατροπαράδοτος πυρήνας 8051 ο οποίος δεν έχει μείνει όπως ήταν αρχικά. Είναι σε θέση να εκτελεί τις εντολές σε πολύ λιγότερες περιόδους του σήματος χρονισμού, ενώ από πλευράς περιφερειακών υποστηρίζει ένα επιπλέον UART, μνήμη προγράμματος 4 έως 32 Kbyte όπως επίσης και ένα κύκλωμα φόρτωσης και εγγραφής της παραπάνω μνήμης μέσω της σειριακής θύρας RS232 οποιουδήποτε υπολογιστή. Για την μεταφορά του προγράμματος χρειάζεται η τοποθέτηση στις κατάλληλες θέσεις της πλακέτας δύο βραχυκυκλωτήρων. Από τη στιγμή εκείνη και μετά τα πάντα ελέγχονται μέσω της σειριακής θύρας του υπολογιστή. Το καλώδιο που χρησιμοποιείται είναι μηδενικού μόντεμ όπως φαίνεται στο κάτω σχήμα

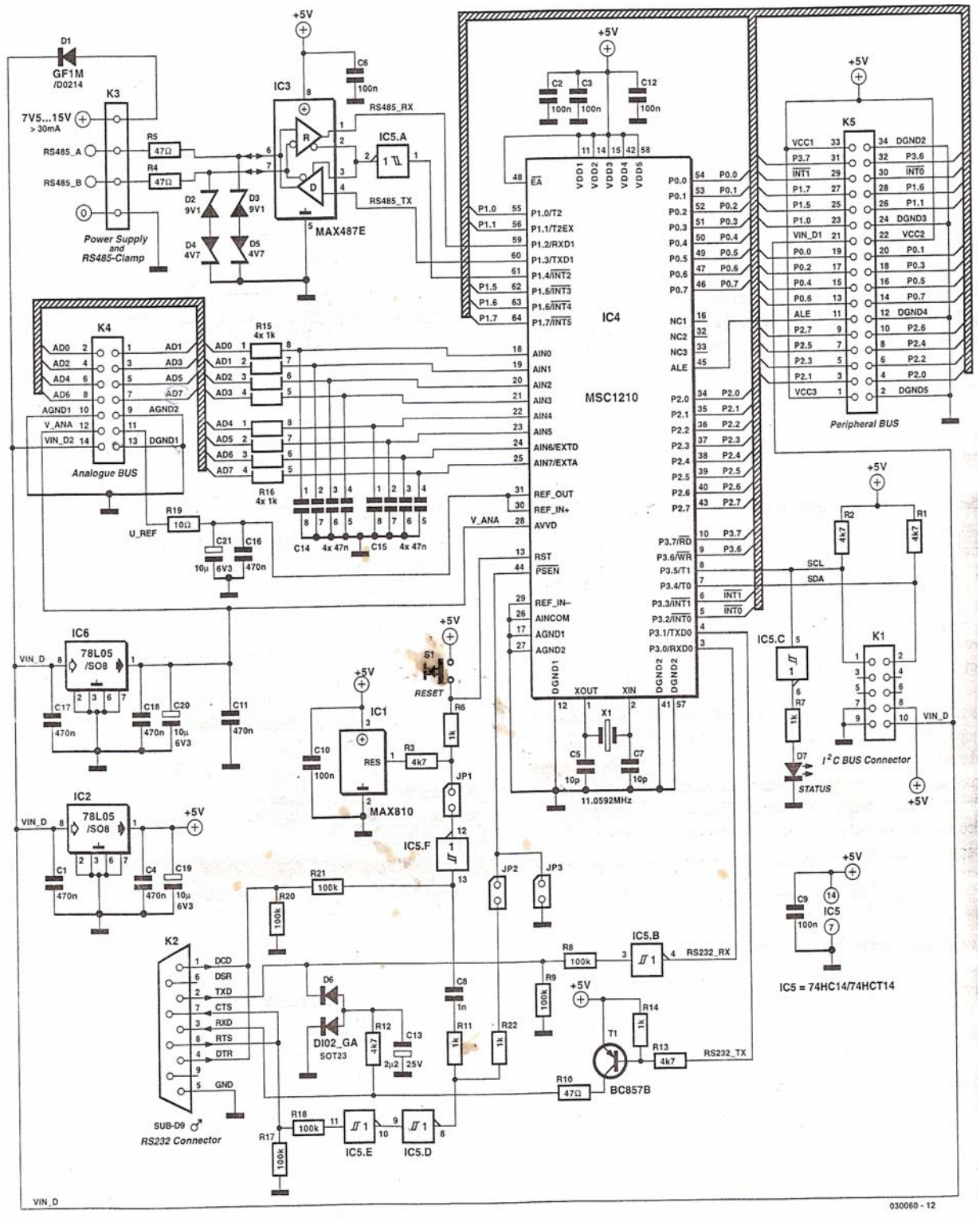


Οι συνδετήρες που έχουν τοποθετηθεί και χρησιμοποιούνται πάνω στις πλακέτες του MSC1210 είναι και από τις δύο πλευρές θηλυκοί. Επιπρόσθετα, φροντίζει να συστρέφει στο εσωτερικό του τα σήματα που εμφανίζονται στις ακίδες 1, 4, 2 και 3 έτσι ώστε τα σήματα εκπομπής που αναδεικνύονται στη μια μεριά να καταλήγουν στην ακίδα λήψης της άλλης και αντιστρόφως. Οι ακίδες 1 και 6 καθενός συνδετήρα συνδέονται μεταξύ τους. Το ένθετο σχήμα δείχνει με παραστατικό τρόπο τα παραπάνω. Οι αριθμοί που σημειώνονται είναι αυτοί που προσδιορίζουν τον αύξοντα αριθμό των ακίδων του συνδετήρα όταν τον κοιτάμε από εμπρός. Εύκολα μπορούμε να ελέγξουμε το καλώδιο αν είναι μηδενικού τύπου χρησιμοποιώντας ένα ωμόμετρο

ή έναν ελεγκτή συνέχειας αγωγών. Όταν κατέβει το πρόγραμμα αφαιρούμε τους βραχυκυκλωτήρες και ο κώδικας παραμένει στην Flash EPROM για 100 χρόνια τουλάχιστον (σύμφωνα με την Texas Instruments..!) έτοιμος να εκτελεστεί όποτε χρειαστεί.

1.1 Ο πυρήνας και τα περιφερειακά του:

Το κυκλωματικό διάγραμμα της πλακέτας μετρήσεων υψηλής ακρίβειας φαίνεται παρακάτω



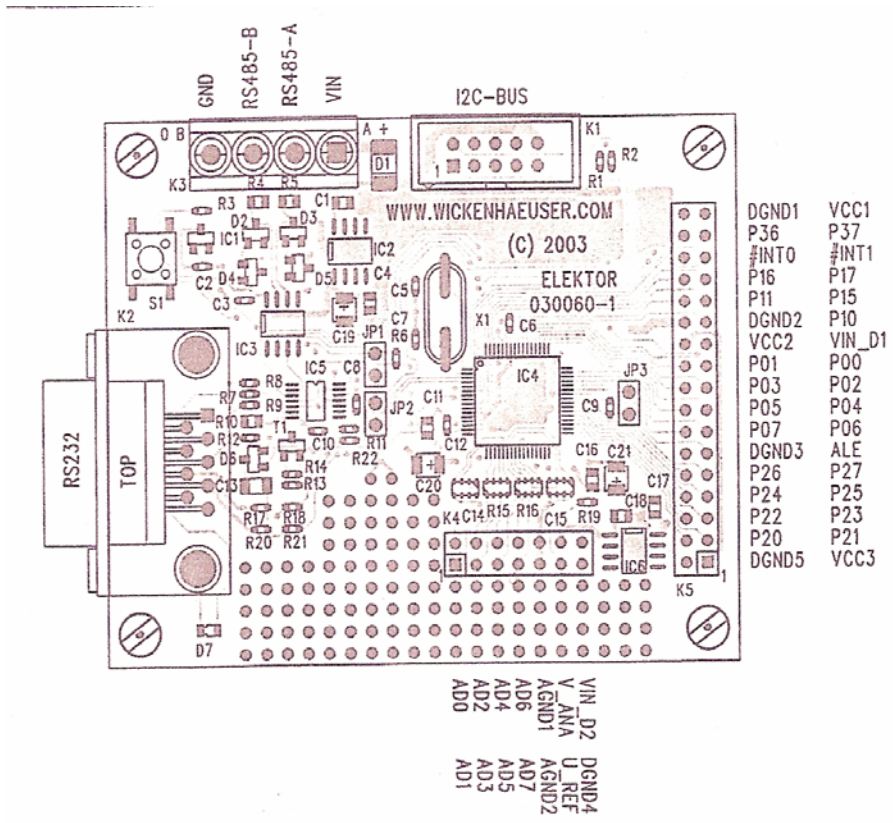
Το βασικότερο εξάρτημα είναι ο μικροελεγκτής MSC 1210 (IC4), ο οποίος τροφοδοτείται δύο διαφορετικούς σταθεροποιητές. IC2 είναι υπεύθυνος για την παροχή τάσης τροφοδοσίας στις ψηφιακές βαθμίδες του, ένω ο IC6 για την παροχή

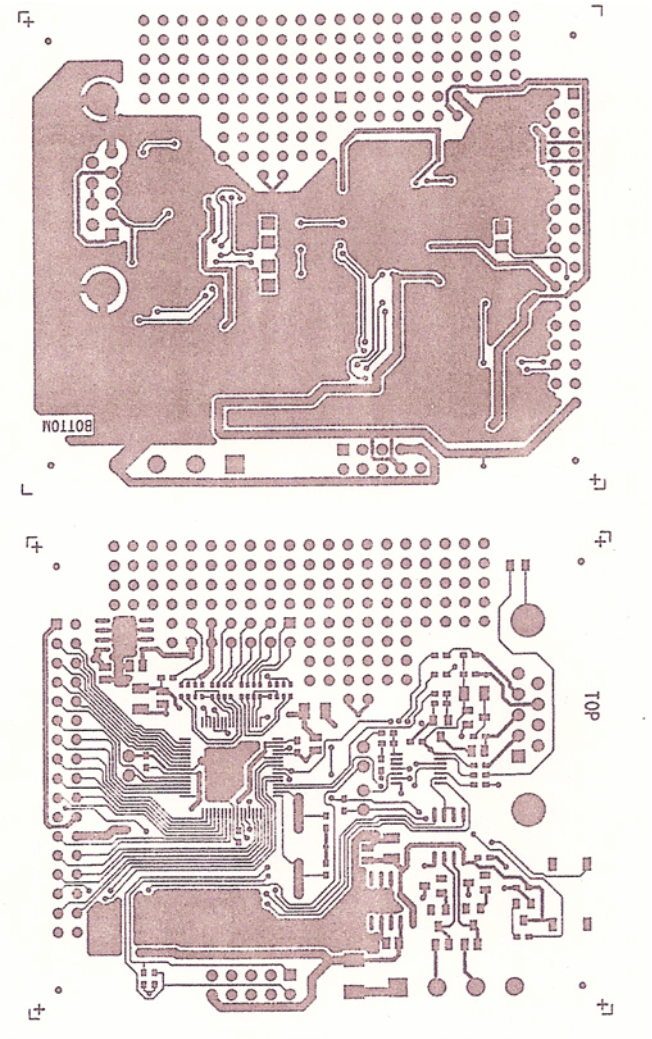
τάσης στις αναλογικές. Η παρουσία δύο σταθεροποιητών αποτελεί επιτακτική ανάγκη, αν θέλουμε να αξιοποιήσουμε στο μέγιστο δυνατόν την υψηλή ακρίβεια του ενσωματωμένου μετατροπέα. Για την αποφυγή θορύβων των γραμμών τροφοδοσίας συμβάλουν κατά πολύ και οι πυκνωτές απόζευξης που τοποθετούνται σε συγκεκριμένα σημεία της πλακέτας. Το ίδιο χρήσιμοι αποδεικνύονται και οι κατάλληλα σχεδιασμένοι χαλκοδιάδρομοι που θωρακίζουν τα ευαίσθητα σημεία της πλακέτας. Ο πυρήνας του μικροελεγκτή χρονίζεται με την βοήθεια ενός κρυστάλλου συχνότητας 11.0592 MHz, που τον κάνει να τρέχει με την υπερδιπλάσια ταχύτητα από ότι ένας συνηθισμένος 8051.(το πόσο ακριβώς ταχύτερος είναι, εξαρτάται από τις εκτελούμενες εντολές). Η αιτία της αυξημένης ταχύτητας οφείλεται στη διαφορετική μηχανή εκτέλεσης των εντολών, που είναι κατά πολύ γρηγορότερη του αρχικού 8051. Αν εφαρμόσουμε στον MSC 1210 σήμα 33 MHz, τότε ο MSC 1210 ισοδυναμεί με ένα συνηθισμένο 8051 χρονισμένο στους 82,5 MHz.Όλες οι αναλογικές είσοδοι προστατεύονται κατά μια έννοια, από στοιχειώδη φίλτρα RC. Ακίδα παροχής τάσης αναφοράς του μικροελεγκτή τροφοδοτεί τα υπόλοιπα κυκλώματα μέσω της αντίστασης R19. Η ικανότητα παροχής ρεύματος από αυτήν φθάνει τα 8mA. Επειδή το ρεύμα αυτό προκαλεί πτώση τάσης στα άκρα της R19, είναι σκόπιμο να αντισταθμίζεται υπολογιστικά η πτώση προκειμένου να ελαχιστοποιούνται τα σφάλματα των μετρήσεων. Η τιμή της τάσης αναφοράς καθορίζεται με την βοήθεια του λογισμικού. Περισσότερα όμως γι' αυτήν θα πούμε στην επόμενη ενότητα. Για την επικοινωνία της πλακέτας με τον υπολογιστή που αναπτύχθηκε το πρόγραμμα της εφαρμογής χρησιμοποιείται το ένα από τα δύο UART του μικροελεγκτή. Κάτω από κανονικές συνθήκες, θα περιμέναμε να δούμε στις ακίδες TxD και RxD του μικροελεγκτή, το καταξιωμένο MAX232 που είναι το πλέον συνηθισμένο εξάρτημα υποστήριξης σειριακών διασυνδέσεων. Παρ' όλα αυτά κάναμε κάτι πιο έξυπνο. Αντί να παράγουμε την απαραίτητη αρνητική τάση βάσει της οποίας θα οδηγούμε τις ακίδες εισόδου του PC, την κλέβουμε με τη βοήθεια του T1 και του IC5.B από μια ακίδα εξόδου του, για να την επιστρέψουμε πάλι σε αυτόν μέσω της ακίδας RxD. Η βαθμίδα του ενσωματωμένου φορτωτή προγράμματος απαιτεί για τη λειτουργία της δύο βοηθητικά σήματα που παράγονται με την βοήθεια των IC5.D, IC5.E και IC5.F. Τα σήματα αυτά είναι ενεργά μόνο όταν έχουν τοποθετηθεί οι βραχυκυκλωτήρες JP1 και JP2. Από αυτούς , ο JP1 επιτρέπει την επανεκκίνηση του MSC 1210 από τον ίδιο PC μέσω της σειριακής θύρας, ενώ ο JP2 αναγκάζει τον μικροελεγκτή να περιμένει το κατέβασμα αμέσως μετά την εκκίνηση του. Η τελευταία πύλη του ολοκληρωμένου, η IC5.C, χρησιμεύει για το άναμμα ενός ενδεικτικού LED. Η δεύτερη σειριακή θύρα του MSC 1210 είναι συμβατή με το πρωτόκολλο RS 485. Αυτό σημαίνει πως η πλακέτα μας μπορεί να συνδεθεί με δίκτυα τέτοιου είδους, των οποίων οι σταθμοί απέχουν μεταξύ τους μέχρι και τα δύο χιλιόμετρα. Το σύνολο των σταθμών εξαρτάται από την ικανότητα παροχής ρεύματος του IC3 και θεωρητικά μπορεί να κυμαίνεται από 32 έως 256. Οι δίοδοι ζένερ που έχουν τοποθετηθεί στο ζεύγος των γραμμών του δικτύου έχουν σαν σκοπό την προστασία των γραμμών από πιθανές αιχμές τάσεων, που αναπτύσσονται σχεδόν πάνω σε καλώδια μεγάλου μήκους. Περισσότερα για τα δίκτυα RS 485 αναφερόμαστε πιο κάτω σε ολόκληρο κεφάλαιο. Σαν τελευταία βαθμίδα έχουμε να να αφαιρούμε αυτήν που είναι υπεύθυνη για την δημιουργία του απαραίτητου διαύλου επικοινωνίας. Επειδή ο MSC 1210 απαιτεί την παρουσία ενός εξειδικευμένου κυκλώματος οδήγησης, αυτό το είδος διαύλου μπορεί να αξιοποιηθεί μόνο όταν ο μικροελεγκτής εργάζεται σαν κύριος (Master) των διακινούμενων πληροφοριών. Όλα τα αναλογικά σήματα που προορίζονται για το μικροελγκτή εμφανίζονται ταυτόχρονα και στις ακίδες του K4. Από εκεί μπορούμε να τα πάρουμε και να τα οδηγήσουμε σε οποιοδήποτε σημείο του τμήματος γενικής πλακέτας, όπου

μπορούμε να έχουμε συναρμολογήσει το δικό μας κύκλωμα. Το σύνολο των ψηφιακών γραμμών του μικροελεγκτή οδηγείται στο δικό του συνδετήρα, τον K5. Αξιοποιώντας τα σήματα που μεταφέρουν οι ακίδες του, μπορούμε να συνδέσουμε στο κύκλωμα πολλές βοηθητικές μονάδες όπως μνήμες κ.τ.λ. Αν συναρμολογήσουμε κυκλώματα της δεύτερης κατηγορίας, θα διαπιστώσουμε πως η κατασκευή μας είναι σε θέση να στέλνει e-mail στα οποία θα περιέχονται τα αποτελέσματα των μετρήσεων που έχει συλλέξει μέσα σε ενα συγκεκριμένο χρονικό διάστημα. Θα μπορεί δηλαδή να συμπεριφέρεται σαν ένα είδος καταγραφικού, το οποίο θα ενημερώνει μέσω του δικτύου, για τα στοιχεία που συνέλεξε.

Στις ακίδες VIN_D1 και VIN_D2 των K4 και K5 εμφανίζεται η τάση τροφοδοσίας των κυκλωμάτων. Θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί μιας που η συγκεκριμένη τάση δεν πρεπει να είναι ποτέ μικρότερη από τα 0 V αλλά ούτε μεγαλύτερη +5V.

Το τυπομένο κύκλωμα φαίνεται στο πιο κάτω σχήμα





1.2 Οδηγός Ethernet CS8900A:

Όπως συνηθίζεται στις κάρτες δικτύου, υπάρχουν δύο LED D1 και D2 για την απεικόνιση της κατάστασης της σύνδεσης με το δίκτυο. Η δίοδος LED D1 αναβοσβήνει για 6 msec κάθε φορά που εκπέμπεται ή λαμβάνεται ένα πακέτο δεδομένων, ή όταν εκδηλωθεί σύγκρουση (collision) μεταξύ δύο πακέτων. Η δεύτερη LED D2 καταδεικνύει αν ο οδηγός CS8900A λαμβάνει τους σωστούς παλμούς. Οι παλμοί αυτοί χρησιμοποιούνται σε ένα δίκτυο Ethernet για το συγχρονισμό μεταξύ πομπών και δεκτών και η δίοδος D2 θα φωτοβολεί μόνον όταν ο παραπάνω συγχρονισμός είναι επιτυχής.

Το ολοκληρωμένο κύκλωμα του δικτύου περιλαμβάνει έναν πλήρη πομποδέκτη 10Base-T. Το 10Base-T αποτελεί ένα πρότυπο επικοινωνίας δικτύου Ethernet στα 10 Mbits/s μέσω συνεστραμένου καλωδίου. Το κύκλωμα απαιτεί λίγα εξωτερικά εξαρτήματα. Ο μετασχηματιστής που βρίσκεται ακριβώς μπροστά από τον συνδετήρα RJ 45 εξασφαλίζει την ηλεκτρική απομόνωση του κυκλώματος.

Το κυκλωμένο κύκλωμα περιλαμβάνει χώρο για πειραματικές συνδέσεις (prototyping area) παρέχοντας με τον τρόπο αυτόν στον χρήστη την δυνατότητα ανάπτυξης εφαρμογών πάνω στην ίδια την πλακέτα, εκτός από τον ήδη διαθέσιμο χώρο της κάρτας του MSC 1210. Στην αριστερη πλευρά του χώρου των ελεύθερων συνδέσεων συναντούμε μερικές εφεδρικές γραμμές σημάτων. Στην κάρτα του τοπικού δικτύου (LAN) βρίσκονται ακόμη δύο επιπλέον δίοδοι LED και ένας πιεστικός διακόπτης. Η τοποθέτηση του συνδετήρα για την ένωση με την μητρική κάρτα, επιτρέπει στην κάρτα δικτύου είτε να τοποθετηθεί κοντά στην μητρική κάρτα είτε κάτω από αυτήν. Στην δεύτερη αυτή περίπτωση οι δύο κάρτες μπορούν να στηριχθούν μαζί σε ασφαλή απόσταση σχηματίζοντας ένα ενιαίο σύστημα. Αν και ο σχεδιασμός του κυκλώματος αυτού είναι αρκετά απλος, υπάρχει κάτι θα πρέπει να τονιστεί. Η κατανάλωση ρεύματος του ολοκληρωμένου οδηγού του δικτύου LAN κυμαίνεται μεταξύ 100 και 120 mA, κατανάλωση σχετικά υψηλή σε σύγκριση με εκείνη του μικροελεγκτή. Η τάση τροφοδοσίας των 5 V για την κάρτα δικτύου λαμβάνεται από την κάρτα του MSC 1210. Για να αποφύγουμε την υπερθέρμανση του σταθεροποιητή της πλακέτας αυτής πρέπει απαραίτητα ολόκληρο το σύστημα των δύο κυκλωμάτων να τροφοδοτηθεί μια πηγή τάσης μετξύ 7.5 και 9 V, αλλά σε καμία περίπτωση μεγαλύτερης τάσης. Το πιο κάτω κύκλωμα είναι το τυπωμένο της κάρτας δικτύου

1.3 Ρυθμίσεις της κάρτας:

Ο Micro Web Server μπορεί να λειτουργήσει μόνον σε ένα δίκτυο TCP / IP. Όπως κάθε υπολογιστής που συνδέεται σε ένα δίκτυο TCP / IP, έτσι και στον μικροελεγκτή εκχωρείται μια μοναδική διεύθυνση, η οποία είναι διεύθυνση IP. Η διεύθυνση που καταχωρήσαμε είναι η *195.251.122.35* .Μετά εγκαταστήσαμε το μC/51 Compiler και στη συνέχεια χρησιμοποιήσαμε το πρόγραμμα Make Wiz για την δημιουργία του περιβάλλοντος εργασίας. Εντός του προγράμματος Make Wiz ανοίξαμε το αρχείο …\SRC\MSC1210\ELM_FLEX\ELM_FLEX.MAK. Έχοντας ανοίξει το αρχείο αυτό, ακολούθως τροποποιήσαμε κάτι πάνω στο κείμενο έτσι ώστε να ενεργοποιηθεί το κουμπί αποθήκευσης(Save). Ενεργοποιήσαμε το πλαίσιο επιλογής (Check Box) με ετικέταWrite JFE Workspace File και στην συνέχεια αποθηκεύσαμε το αρχείο. Αμέσως μετά εκκινήσαμε το συντάκτη JFE. Στο περιβάλλον του JFE εκτελέσαμε την εντολή Open workspace για να ανοίξουμε το αρχείο...\SRC\MSC1210\ELM_FLEX\ELM_FLEX.WSP. Όλα τα αρχεία που ανήκουν στην συγκεκριμένη εφαρμογή θα εμφανίστηκαν στο παράθυρο του συντάκτη. Στο σημείο αυτό βάλαμε και την IP στον κώδικα μας στη γραμμή COMPOSE-IP (my_ip,195.251.122.35)

Το περιβάλλον εργασίας που δημιουργήθηκε με το Make Wiz προκαλεί την εμφάνιση τριών κουμπιών στον JFE : ‘MAKE’ ‘RE-MAKE’ ‘DL.BAT’. Με το κουμπί MAKE εκτελείται η μεταγλώττιση της εφαρμογής αλλά με τον περιορισμό ότι στην διαδικασία αυτή λαμβάνουν μέρος μόνον όσα αρχεία έχουν προηγούμενα τροποποιηθεί. Αυτός είναι ασυνήθης και πιο γρήγορος τρόπος για να δημιουργηθεί το δεκαεξαδικό αρχείο που θα φορτωθεί στον μικροελεγκτή. Το κουμπί RE-MAKE χρησιμοποιείται όταν έχει μεταβληθεί κάποιο αρχείο που δεν εμφανίζεται στο περιβάλλον εργασίας, όπως για παράδειγμα ένα αρχείο επικεφαλίδας. Με την εκτέλεση αυτής της εντολής επαναμεταγλωτίζονται όλα τα αρχεία που αποτελούν τη συγκεκριμένη εφαρμογή στο σύνολο της. Τέλος το κουμπί DL.BAT αποστέλλεται το δεκαεξαδικό αρχείο που δημιουργήθηκε στην κάρτα MSC. Η διαδικασία αυτή

αντιστοιχεί ουσιαστικά στην εκτέλεση ενός τυπικού αρχείου δέσμης στο οποίο ο συντάκτης JFE έχει περάσει μια παράμετρο. Η παράμετρος αυτή είναι πάντοτε το όνομα του αρχείου που πρόκειται να φορτωθεί στο υλικό της κάρτας και το οποίο στη συγκεκριμένη περίπτωση είναι το ELM_FLEX.

Στο αρχείο δέσμης εμφανίζεται η συγκεκριμένη εντολή με την οποία εκκινείται η διαδικασία φόρτωσης στην κάρτα MSC Στη συγκεκριμένη περίπτωση η εντολή αυτή είναι, download /F%1.hex / X11 / P1 / T / B9600. Η παράμετρος P1 ορίζει ότι για την διαδικασία προγραμματισμού θα πρέπει να χρησιμοποιηθεί η σειριακή θύρα COM1 του υπολογιστή. Αν είναι απαραίτητο μπορούμε την ρύθμιση αυτή να την αλλάξουμε. Γεφυρώσαμε τους J1 και J2 πατήσαμε το κουμπί DL.BAT και κατέβηκε το πρόγραμμα στον μικροελγκτή. Στο τέλος της διαδικασίας και αφού όλα είχαν πάει μια χαρά μας επέστρεψε ένα μήνυμα χαιρετισμού μέσα από το ίδιο το πρόγραμμα και αφαιρέσαμε και τους J1 και J2 έτσι ώστε να μην μπορεί να πειραχτεί το πρόγραμμα από καμία εξωτερική αιτία.

1.4 Σύνδεση σε λειτουργία On – Line:

Από την στιγμή που ενεργοποιήθηκε ο server ξεκινήσαμε την διαδικασία ελέγχου. Συνδέσαμε την πλακέτα στο δίκτυο , ανιχνεύτηκε σήμα δικτύου Ethernet και η LED 2 άρχισε να φωτοβολεί συνεχόμενα. Στη συνέχεια κάναμε Ping μέσω του MS DOS (ping 195.251.122.35) την διεύθυνση μας, αυτή μας απάντησε η απόκριση του server και άρχισε να αναβοσβήνει η LED 1 ως ένδειξη μεταφοράς δεδομένων μέσω δικτύου Ethernet. Η διαδικασία Ping αποτελεί ένα απλό πρωτόκολλο με το οποίο επιτρέπεται η εκπομπή μερικών bytes ενώ στη συνέχεια το σύστημα αναμένει να λάβει την ηχώ τους. Πρόκειται για έναν απλό και πολύ πρακτικό τρόπο ελέγχου μιας σύνδεσης δικτύου. Στη συνέχεια πήγαμε σε ένα Web Browser πληκτρολογήσαμε την διεύθυνση http://195.251.122.35 και έφτασε η σελίδα html στην οθόνη του υπολογιστή μας μέσω του Micro Web Server.

1.5 Λειτουργία:

Τι συμβαίνει όταν αιτούμαστε μια σελίδα; Αρχικά εκτελούμε την σύνδεση με μια διεύθυνση IP. Βέβαια είναι πιο περίπλοκη η διαδικασία, για την ακρίβεια εκτελούμε την σύνδεση με μια υποδοχή (socket) σε μια συγκεκριμένη διεύθυνση. Μια υποδοχή θα λέγαμε ένα είδος συνδετήρα ο οποίος στην προκειμένη περίπτωση είναι κατάλληλος μόνον για συνδέσεις με ιστιοσελίδες. Σε κάθε υποδοχή εκχωρείται και ένας συγκεκριμένος αριθμός θύρας. Η θύρα 80 (port 80) χρησιμοποιείται συχνά για τις εφαρμογές Web Server. Μπορούμε να το δούμε αυτό στην γραμμή προγράμματος SOCKET_SETUP(!, SOCKET_TCP, 80, FLAG_PASSIVE_OPEN). Η τελευταία παράμετρος φανερώνει ότι η συγκεκριμένη υποδοχή είναι παθητικού τύπου, δηλαδή βρίσκεται σε κατάσταση αναμονής αιτήσεων από εξυπηρετούμενες εφαρμογές (client). Η οντότητα μιας υποδοχής δημιουργείται με ένα επαναληπτικό βρόγχο FOR. Το πλήθος των υποδοχών που δημιουργούνται καθορίζει το πλήθος των εξυπηρετούμενων εφαρμογών που μπορούν να συνδεθούν ταυτόχρονα με το συγκεκριμένο server. Επειδή κάθε μια υποδοχή παρουσιάζει κάποια απαίτηση σε μνήμη, ο συνολικός του αριθμός είναι περιορισμένος. Το ολοκληρωμένο CS8900A που χρησιμοποιείται στην εφαρμογή μας διαθέτει ενδιάμεση μνήμη (buffer) -

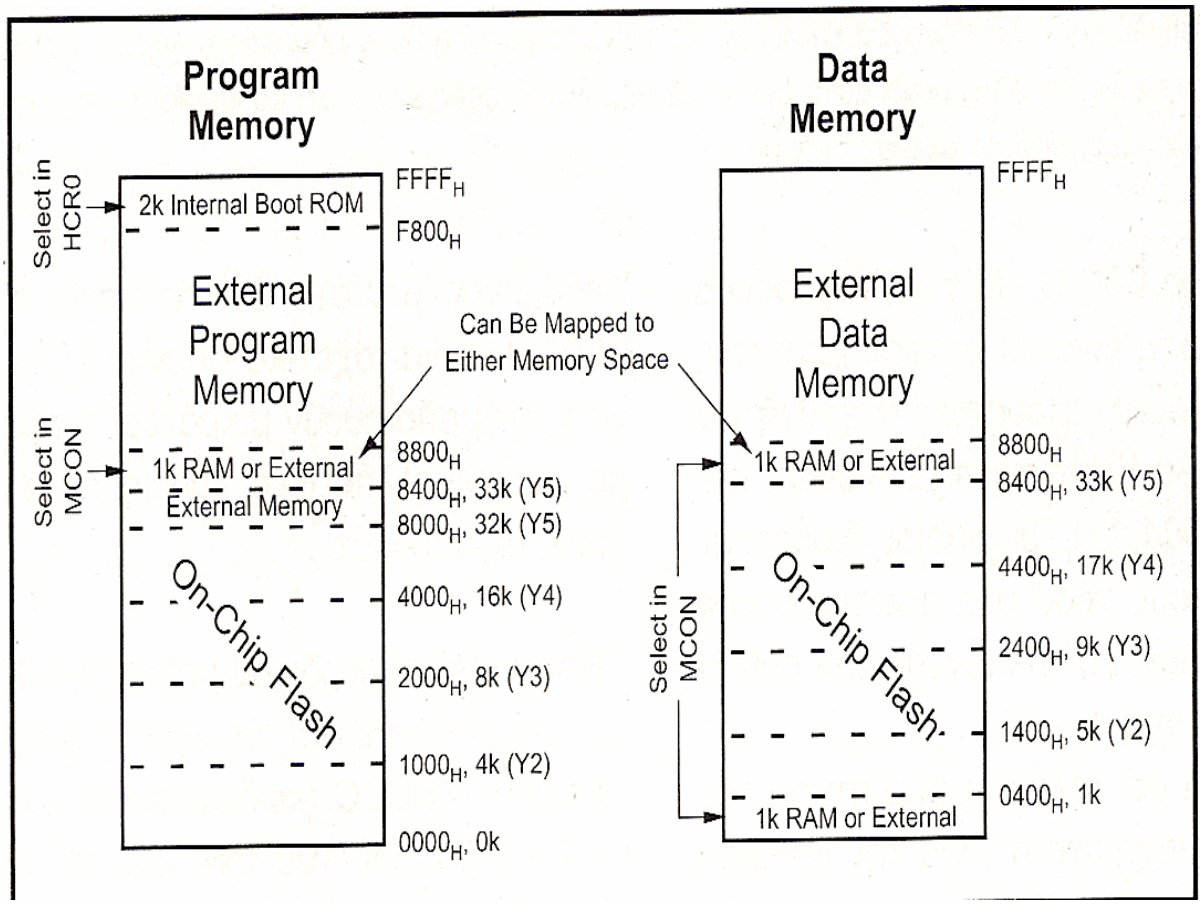
(περίπου 4Kbyte) – για την προσωρινή αποθήκευση των εισερχόμενων πακέτων δικτύου Ethernet. Η χωρητικότητα μνήμης αυτή δεν θεωρείται ιδιαίτερα μεγάλη στην περίπτωση που αρκετοί χρήστες επιχειρούν να συνδεθούν ταυτόχρονα με τον server, ή όταν γίνεται αίτηση για αντικείμενα δεδομένων μεγάλου μεγέθους, όπως είναι οι εικόνες. Στην πραγματικότητας αυτό δεν αποτελεί σοβαρό πρόβλημα εφόσον σύμφωνα με το πρωτόκολλο TCP επιτρέπεται να μην απαντηθεί κάποιο τυχαίο πακέτο δεδομένων. Αν είναι απαραίτητο η εξυπηρετούμενη εφαρμογή (client) αποστέλλει τα αναπάντητα πακέτα με δική της πρωτοβουλία.

Αφού δημιουργηθούν οι υποδοχές, το πρόγραμμα ELM_FLEX.C εκχωρεί αρχικές τιμές στο τμήμα που αφορά τον μετατροπέα A / D του μικροελεγκτή, μερικές γραμμές του κώδικα πιο κάτω. Στη συνέχεια το πρόγραμμα εισέρχεται σε ένα βρόγχο FOR ο οποίος επαναλαμβάνεται συνεχώς. Εντός του βρόγχου αυτού καλείται περιοδικά η ρουτίνα poll_webserver(). Όσο το αποτέλεσμα που επιστρέφεται μετά από κάθε εκτέλεση της ρουτίνας αυτής είναι ‘0’, μπορούν να εκτελούνται και άλλες ρουτίνες στα πλαίσια του βρόγχου. Είναι πολύ σημαντικό ωστόσο να εξασφαλίζεται το γεγονός ότι οι ρουτίνες των χρηστών δεν θα απασχολούν για μεγάλο χρονικό διάστημα τον επεξεργαστή, διότι διαφορετικά η πρόσβαση στον webserver καθίσταται αδύνατη.

Η στοίβα FlexGate TCP/IP λειτουργεί με βάση τον μηχανισμό των γεγονότων (events). Ο Micro Web Server αποκρίνεται μόνον στο EVENT_HTTP_REQUEST (αίτηση σελίδας) και στο EVENT_SOCKET_IDLETIMER (το οποίο έχει περίοδο περίπου 0,5 s). Αν σε μία εφαρμογή client αιτηθεί πρόσβαση σε κάποια σελίδα, πρώτα ζητείται το όνομα της σελίδας με την βοήθεια της webpage_name(). Στη συνέχεια με χρήση της webpage_blind() η αντίστοιχη σελίδα προετοιμάζεται για απόκριση. Σελίδες οι οποίες πρόκειται να είναι εξωτερικά διαθέσιμες θα πρέπει να δηλωθούν προκαταβολικά ως array extern code uchar. Η διαδικασία με τον τρόπο αυτό ολοκληρώνεται, αν η ζητούμενη σελίδα δεν περιέχει κάποια δυναμικά δεδομένα. Βέβαια η πραγματική αξία της μικρής και χρήσιμης αυτής διάταξης, βασίζεται ακριβώς στην φύση τρων δυναμικών δεδομένων. Ένα παράδειγμα δυναμικών δεδομένων αποτελούν τα δεδομένα των μετρήσεων που προέρχονται από την κάρτα του μικροελεγκτή. Τέτοιου είδους δεδομένα μπορούν εύκολα να ενσωματωθούν σε μια ιστοσελίδα. Και βέβαια, από την άλλη πλευρά μπορείτε να τηλεχειριστείτε τις εξόδους της κάρτας του μικροελεγκτή μέσω μιας ιστοσελίδας.

2. Μνήμη Flash του μικροελεγκτή MSC 1210

Η οικογένεια MSC 1210 έχουν την ίδια δομή μνήμης με τους 8051. Διαθέτουν δηλαδή, την ίδια μνήμη προγράμματος (μέχρι 64 Kbyte) που προσπελαύνονται με τον παραδοσιακό τρόπο των μικροελεγκτών της *Intel*. Αυτό σημαίνει πως όλοι οι MSC 1210 ασπάζονται την αρχιτεκτονική Harvard που χαρτογραφεί την μνήμη δεδομένων σε διαφορετικό χώρο από την μνήμη προγράμματος όπως φαίνεται στο σχήμα που ακολουθεί



Αν όμως μελετήσουμε καλύτερα τις ιδιαιτερότητες του πυρήνα των νέων μικροελεγκτών θα δούμε πως αφήνουν πολλά περιθώρια αλληλοεπικαλύψεων. Σε μια τέτοια παρατυπία καταφεύγουμε μάλλον σπάνια, αλλά στην περίπτωση που χρειαστεί μας επιτρέπει π.χ. να τοποθετήσουμε την εσωτερική RAM του 1 Kbyte είτε στο χάρτη της μνήμης προγράμματος είτε στο χάρτη της μνήμης δεδομένων. Το ίδιο άλλωστε μπορεί να γίνει και με τμήματα της εξωτερικής μνήμης δεδομένων.

2.1 Διαμερίσματα στη μνήμη Flash:

Η δημιουργία διαμερισμάτων μέσα στη μνήμη Flash επινοήθηκε από την Texas Instruments και ονομάσθηκε, από αυτήν Flash Partitioning. Στην πράξη υποδουλώνει τη δυνατότητα του μικροελεγκή να δεσμεύει ένα συγκεκριμένο μέρος της διαθέσιμης μνήμης Flash για την αποθήκευση του προγράμματος αφήνοντας το υπόλοιπο να συμπεριφέρεται σαν μνήμης δεδομένων. Ο καθορισμός αυτός κατά την φάση της μεταφοράς του προγράμματος του χρήστη από τον υπολογιστή που το ανέπτυξε στο μικροελεγκτή. Ο κατασκευαστής επιτρέπει, ανάλογα με τον τύπο του μικροελεγκτή, την ανάδειξη διαμερισμάτων με ένα από τους επτά διαφορετικούς τρόπους

HCR0	MSC1210Y2		MSC1210Y3		MSC1210Y4		MSC1210Y5	
DFSEL	PM	DM	PM	DM	PM	DM	PM	DM
000	0kB	4kB	0kB	8kB	—	—	—	—
001	0kB	4kB	0kB	8kB	—	—	0kB	32kB
010	0kB	4kB	0kB	8kB	0kB	16kB	16kB	16kB
011	0kB	4kB	0kB	8kB	8kB	8kB	24kB	8kB
100	0kB	4kB	4kB	4kB	12kB	4kB	28kB	4kB
101	2kB	2kB	6kB	2kB	14kB	2kB	30kB	2kB
110	3kB	1kB	7kB	1kB	15kB	1kB	31kB	1kB
111 (default)	4kB	0kB	8kB	0kB	16kB	0kB	32kB	0kB

NOTE: When a 0kB program memory configuration is selected program execution is external. "—" is reserved.

Αν θέλουμε να γίνουμε πιο συγκεκριμένοι θα λέγαμε πως ο καθορισμός των διαμερισμάτων γίνεται κατά τη φάση της της συγγραφής του προγράμματος και μάλιστα στις πρώτες γραμμές κώδικα του. Από ότι γνωρίζουμε, οι μοναδικοί μικροελεγκτές που μπορούν συμπεριφερθούν με αυτό τον τρόπο είναι οι MSC 1210 της Texas Instruments. Ο τεμαχισμός της μνήμης προγράμματος ή της μνήμης δεδομένων εξασφαλίζει ότι οι προσπελάσεις σε εσφαλμένες διευθύνσεις της περιοχής δεδομένων δεν θα έχουν καμία συνέπεια στην ακεραιότητα του προγράμματος που έχει ήδη κατέβει στον μικροελεγκτή. Η συμπεριφορά αυτή δημιουργεί νέα πρότυπα σε ότι αφορά την ασφάλεια και την αξιοπιστία των συστημάτων μικροελεγκτών. Ακόμα το γεγονός ότι τα τμήματα της μνήμης που φιλοξενούν δεδομένα βρίσκονται σε συγκεκριμένες περιοχές του χάρτη μνήμης όπως βλέπουμε στο σχήμα

HCR0	MSC1210Y2		MSC1210Y3		MSC1210Y4		MSC1210Y5	
DFSEL	PM	DM	PM	DM	PM	DM	PM	DM
000 (reserved)	—	—	—	—	—	—	—	—
001	0000	0400-13FF	0000	0400-23FF	0000	0400-43FF	0000	0400-83FF
010	0000	0400 13FF	0000	0400 23FF	0000	0400 43FF	0000-3FFF	0400-43FF
011	0000	0400-13FF	0000	0400-23FF	0000-1FFF	0400-23FF	0000-5FFF	0400-23FF
100	0000	0400-13FF	0000-0FFF	0400-13FF	0000-2FFF	0400-13FF	0000-6FFF	0400-13FF
101	0000-07FF	0400-0BFF	0000-17FF	0400-0BFF	0000-37FF	0400-0BFF	0000-77FF	0400-0BFF
110	0000-0BFF	0400-07FF	0000-1BFF	0400-07FF	0000-3BFF	0400-07FF	0000-7BFF	0400-07FF
111 (default)	0000-0FFF	0000	0000-1FFF	0000	0000-3FFF	0000	0000-7FFF	0000

NOTE: Program memory accesses above the highest listed address will access external program memory.

έχει σαν αποτέλεσμα την απλοποίηση της σχεδίασης και οργάνωσης του λογισμικού. Στα πλαίσια των παραπάνω δυνατοτήτων εντάσσεται και η δυνατότητα εγγραφής της ίδιας της μνήμης προγράμματος τη στιγμή που η CPU εκτελεί ένα πρόγραμμα που βρίσκεται ήδη γραμμένο σε αυτήν (λειτουργία In Application, IAP). Για την ανάγνωση της μνήμης δεν απαιτείται καμία ειδική λειτουργία . Ο κώδικας που έχει γραφτεί σε αυτήν ξεκινά πάντα από την διεύθυνση 0000H, ενώ τα byte που αποθηκεύονται στη μνήμη δεδομένων από την 0400H και η επιπρόσθετη μνήμη RAM του ενός Kbyte φαίνεται στην περιοχή 0000H – 03FFFH.

2.2 Η εγγραφή της μνήμης Flash

Για την εγγραφή της μνήμης Flash πρέπει να καταφύγουμε σε μερικά τεχνάσματα. Τα προγράμματα που μας χρειάζονται γι αυτήν τη δουλειά φιλοξενούνται στην ROM εκκίνησης που εδρεύει στα 2 ψηλότερα Kbyte της μνήμης προγράμματος. Για να τα αξιοποιήσουμε πρέπει να επισυνάψουμε στο πηγαίο πρόγραμμα γλώσσας C το αρχείο επικεφαλίδας ROM 1210. Το αρχείο αυτό συνοδεύει το πακέτο του μεταγλωττιστή μC/51. Μιλώντας καθαρά τεχνικά είναι σκόπιμο να ξεκαθαρίσουμε στο σημείο αυτό, τις διαφορές μεταξύ των μνημών EPROM και Flash. Σε μία μνήμη EPROM κάθε μία θέση της μπορεί να διαγράφει χωρίς αυτή η κίνηση να έχει επιπτώσεις στις γειτονικές της. Σε μία Flash αυτό είναι

αδύνατο. Κάθε προσπάθεια διαγραφής του περιεχομένου μιας θέσης έχει σαν συνέπεια τη διαγραφή ενός ολόκληρου τομέα (128 byte). Οι διαγραμμένες θέσεις χαρακτηρίζονται από υψηλή στάθμη ‘1’ σε όλα τα ψηφία τους (FFH), ενώ η μετέπειτα εγγραφής τους τροποποιεί στην πραγματικότητα μόνο την κατάσταση εκείνων των ψηφίων που πρέπει να αποκτήσουν την τιμή ‘0’. Ένα από τα πολλά προγράμματα επίδειξης που προσφέρει η Texas Instruments επιτρέπει στη μνήμη Flash να συμπεριφέρεται σαν EPROM. Ο τρόπος που το πετυχαίνει είναι απλός: Μεταφέρει το περιεχόμενο όλου του τομέα του οποίου θέλουμε να αλλάξουμε ένα byte ,στη μνήμη RAM. Τροποποιεί μέσα στη RAM το byte που έχουμε ζητήσει και στη συνέχεια διαγράφει όλο τον τομέα της Flash. Τέλος, επανεγγράφει τα περιεχόμενα της RAM στη Flash. Το αποτέλεσμα είναι η αλλαγή του περιεχομένου μόνο της επιλεγμένης θέσης. Ο καταχωρητής HCR0 είναι εκείνος που φροντίζει για τον τεμαχισμό της Flash σε διαμερίσματα προγράμματος και δεδομένων. Ο καταχωρητής αυτός μπορεί να εγγράφει μόνο κατά τη διάρκεια του κατεβάσματος του προγράμματος εφαρμογής, εν΄ όψη μπορεί να διαβαστεί ανά πάση στιγμή μέσω οποιασδήποτε εντολής ικανής να προσπελάσει τη μνήμη προγράμματος στη θέση 0807 H. Στην πραγματικότητα όλοι οι MSC 1210 διαθέτουν δύο τέτοιους καταχωρητές : Τον HCR0 και τον HCR1. Για την δημιουργία των διαμερισμάτων χρησιμοποιείται, όμως, μόνο ο πρώτος. Τα προγράμματα πηγαίου κώδικα που συνοδεύουν το μεταγλωττιστή μC/ 51 δείχνουν με απλό τρόπο το πως μπορούμε να επέμβουμε στο υλικό του μικροελεγκτή μέσα από το λογισμικό. Μας επιτρέπουν να εισάγουμε τιμές στον HCR0 ή και να δεσμεύσουμε συγκεκριμένες περιοχές της μνήμης.Φυσικά, τόσο η μνήμη προγράμματος όσο και η μνήμη δεδομένων θα μπορούν να διαβάζονται χωρίς περιορισμούς. Εξ’ ίσου σημαντική αποδεικνύεται και η φόρτωση των καταχωρητών χρόνου του MSC 1210 σύμφωνα με την ονομαστική τιμή του χρησιμοποιούμενου κρυστάλλου χρονισμού. Το ίδιο σημαντικό είναι να ορίσουμε και το σωστό ρυθμό μετάδοσης δεδομένων μέσω της σειριακής θύρας. Αν όλα αυτά γίνουν με το σωστό τρόπο, ο μικροελεγκτής θα μπορέσει να φορτωθεί με το πρόγραμμα της εφαρμογής μας και στη συνέχεια να το εκτελέσει. Για το κατέβασμα του προγράμματος εφαρμογής θα χρησιμοποιήσουμε το πρόγραμμα Umshell ενεργοποιώντας την επιλογή RUN.

3. ΤΟ ΠΡΩΤΟΚΟΛΛΟ TCP/IP

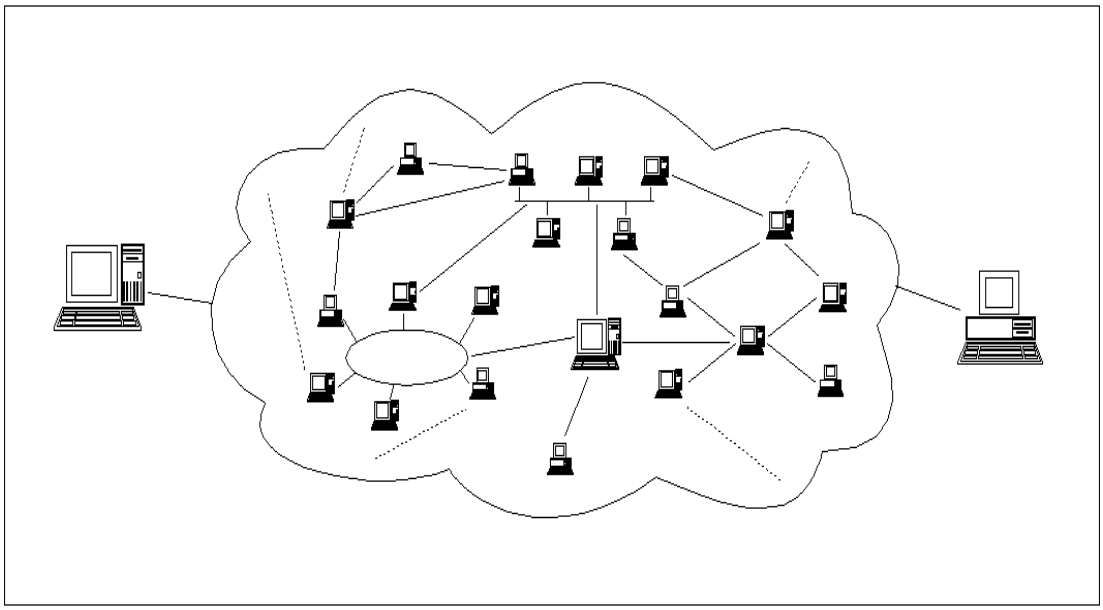
Στην καθημερινή μας ζωή, πρωτόκολλο είναι ένα σύνολο από συμβάσεις που καθορίζουν το πώς πρέπει να πραγματοποιηθεί κάποια διαδικασία. Στον κόσμο των δικτύων, πρωτόκολλο είναι ένα σύνολο από συμβάσεις που καθορίζουν το πώς ανταλλάσσουν μεταξύ τους δεδομένα οι υπολογιστές του δικτύου. Το πρωτόκολλο είναι αυτό που καθορίζει το πώς διακινούνται τα δεδομένα, το πώς γίνεται ο έλεγχος και ο χειρισμός των λαθών, κλπ. Το Internet δεν είναι ένα απλό δίκτυο, αλλά ένα διαδίκτυο. Χρειάζεται επομένως ένα σύνολο από συμβάσεις που να καθορίζουν το πώς ανταλλάσσουν μεταξύ τους δεδομένα υπολογιστές που μπορεί να είναι διαφορετικού τύπου και να ανήκουν σε διαφορετικά δίκτυα.

Ακριβώς αυτό το σύνολο συμβάσεων προσφέρει το TCP/IP. Όλοι οι υπολογιστές που είναι συνδεδεμένοι στα χιλιάδες μικρότερα δίκτυα του Internet

τρέχουν το πρωτόκολλο TCP/IP κι έτσι μιλούν μια κοινή γλώσσα που τους επιτρέπει να συνεννοούνται παρά τις διαφορές τους.

3.1 Τι κάνει το TCP/IP

Ας υποθέσουμε ότι θέλουμε να μεταφέρουμε δεδομένα από έναν υπολογιστή που είναι συνδεδεμένος στο Internet και βρίσκεται π.χ. στην Αμερική, στο MIT, σε έναν άλλον που είναι επίσης συνδεδεμένος στο Internet και βρίσκεται π.χ. στην Ελλάδα, στο Πανεπιστήμιο Θεσσαλίας. Μεταξύ των δύο υπολογιστών παρεμβάλλεται το "σύννεφο" του Internet, δηλ. ένα πλέγμα από συνδέσεις και ενδιάμεσους υπολογιστές.



Οι δύο τελικοί υπολογιστές και το “σύννεφο” του Internet

Το Internet χρησιμοποιεί την τεχνολογία μεταγωγής πακέτων για τη μεταφορά των δεδομένων: τα δεδομένα κόβονται σε κομμάτια που ονομάζονται πακέτα και σε κάθε πακέτο μπαίνει μια “επικεφαλίδα” με τις διευθύνσεις του υπολογιστή - αποστολέα και του υπολογιστή - παραλήπτη. Σημειώνουμε ότι σε κάθε υπολογιστή του Internet αντιστοιχίζεται μία διεύθυνση που ονομάζεται διεύθυνση IP (περισσότερα γι αυτές τις διευθύνσεις στην επόμενη παράγραφο).

Το πρωτόκολλο IP είναι υπεύθυνο για το πέρασμα του πακέτου από υπολογιστή σε υπολογιστή μέσα από το “σύννεφο” των συνδέσεων. Καθώς το IP δρομολογεί το κάθε πακέτο μέσα στο δίκτυο, προσπαθεί να το παραδώσει, αλλά δεν μπορεί να εγγυηθεί ούτε ότι το πακέτο θα φτάσει στον προορισμό του ούτε ότι τα διάφορα πακέτα που αποτελούν τα αρχικά δεδομένα θα φτάσουν με τη σειρά με την οποία στάλθηκαν ούτε ότι το περιεχόμενο των πακέτων θα φτάσει αναλλοίωτο.

Το TCP προσφέρει ένα αξιόπιστο πρωτόκολλο πάνω από το IP. Εγγυάται ότι τα πακέτα θα παραδοθούν στον προορισμό τους, ότι θα φτάσουν με τη σειρά με την οποία στάλθηκαν και ότι τα περιεχόμενα των πακέτων θα φτάσουν αναλλοίωτα (δηλ. όπως στάλθηκαν). Το TCP δουλεύει ως εξής: το κάθε πακέτο δεδομένων αριθμείται. Ο υπολογιστής - παραλήπτης και ο υπολογιστής - αποστολέας, αλλά όχι οι ενδιάμεσοι υπολογιστές, παρακολουθούν τους αριθμούς των πακέτων και ανταλλάσσουν μεταξύ τους πληροφορίες. Ο παραλήπτης λαμβάνει το πρώτο πακέτο, το δεύτερο, κλπ. Σε περίπτωση που παρουσιαστεί κάποιο πρόβλημα στο δίκτυο είτε χαθεί κάποιο πακέτο κατά τη διάρκεια της μετάδοσης, το ξαναζητάει και ο αποστολέας είναι υπεύθυνος για την αναμετάδοση του. Ο παραλήπτης ελέγχει επίσης αν το περιεχόμενο των πακέτων φτάνει σωστά.

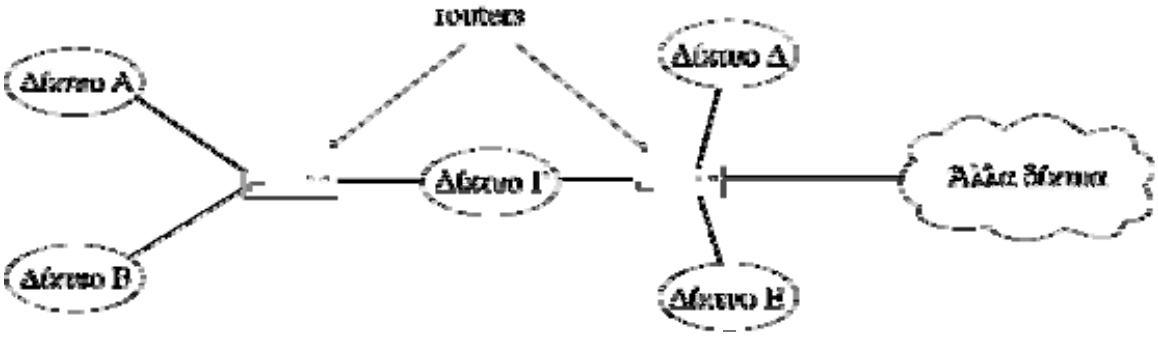
Η μέθοδος αυτή εξασφαλίζει αξιοπιστία και ταχύτητα διότι οι ενδιάμεσοι υπολογιστές δεν εκτελούν ελέγχους.

Τώρα λοιπόν που γνωρίσαμε το TCP/IP μπορούμε να δώσουμε έναν πιο "επίσημο" ορισμό του Internet: ένα δίκτυο αποτελούμενο από δίκτυα υπολογιστών που επικοινωνούν χρησιμοποιώντας το πρωτόκολλο TCP/IP. Όπως θα δούμε παρακάτω, η διαδρομή που ακολουθεί ένα πακέτο μέσα από το "σύννεφο" των συνδέσεων δεν είναι προκαθορισμένη.

3.2 Πώς δρομολογούνται τα πακέτα

Το πρωτόκολλο IP είναι υπεύθυνο για το πέρασμα ενός πακέτου δεδομένων από υπολογιστή σε υπολογιστή. Όλα τα δίκτυα που συνδέονται στο Internet "καταλαβαίνουν" τη γλώσσα IP κι έτσι μπορούν να συνεννοούνται και να ανταλλάσσουν δεδομένα με ομοιόμορφο τρόπο.

Τα δίκτυα του Internet συνδέονται μεταξύ τους με ειδικούς υπολογιστές που ονομάζονται δρομολογητές (routers) ή πύλες (gateways). Ένας router είναι λοιπόν ένας υπολογιστής που συνδέει δύο ή περισσότερα δίκτυα (που μπορεί να είναι διαφορετικού τύπου) και έτσι ανήκει σε δύο ή περισσότερα δίκτυα ταυτόχρονα.



Ένας router συνδέει δύο ή περισσότερα δίκτυα

Η δουλειά των routers είναι να δρομολογούν τα πακέτα των δεδομένων μέσα από τα διάφορα δίκτυα που αποτελούν το Internet μέχρις ότου τα επιδώσουν στον προορισμό τους. Ας δούμε πώς γίνεται αυτό:

Ας θεωρήσουμε πάλι ότι ένας υπολογιστής που βρίσκεται κάπου στο Internet θέλει να στείλει δεδομένα σε κάποιον άλλον υπολογιστή. Τα δεδομένα κόβονται σε πακέτα και το IP που εκτελείται στον υπολογιστή - αποστολέα ετοιμάζεται να στείλει το κάθε πακέτο. Εισάγει λοιπόν στην επικεφαλίδα του πακέτου τις IP διευθύνσεις του αποστολέα και του παραλήπτη και κατόπιν, βάσει των διευθύνσεων αυτών, ελέγχει αν ο παραλήπτης βρίσκεται στο ίδιο δίκτυο με τον αποστολέα.

Εάν ναι, το πακέτο στέλνεται κατευθείαν στον παραλήπτη χωρίς να χρειαστεί να διαβεί τα όρια του δικτύου. Εάν όχι, προωθείται στον router που είναι συνδεμένος

με το δίκτυο. Ο router με τη σειρά του ελέγχει αν ο παραλήπτης βρίσκεται σε κάποιο από τα υπόλοιπα δίκτυα με τα οποία είναι συνδεδεμένος. Εάν ναι, το πακέτο στέλνεται κατευθείαν στον παραλήπτη στο δίκτυο αυτό. Εάν όχι, το πακέτο προωθείται στον επόμενο router, κ.ο.κ. μέχρις ότου το πακέτο προωθηθεί τελικά στον router που είναι συνδεδεμένος στο ίδιο δίκτυο με τον παραλήπτη. Το πακέτο μπορεί έτσι να περάσει από πολλούς routers μέχρις ότου φτάσει στον προορισμό του.

Οι routers διατηρούν πίνακες που προσδιορίζουν την κατεύθυνση που πρέπει να πάρει ένα πακέτο προκειμένου να φτάσει στον προορισμό του. Βάσει αυτών των πινάκων αποφασίζουν ποιος θα είναι ο επόμενος router στον οποίο θα πρέπει να προωθήσουν το πακέτο. Κάθε φορά, το πακέτο μετακινείται όλο και πιο κοντά προς τον προορισμό του έως ότου τελικά τον φτάσει.

Ένα μεγάλο πλεονέκτημα αυτής της μεθόδου είναι ότι η διαδρομή που ακολουθεί ένα πακέτο δεν είναι προκαθορισμένη, αλλά επιλέγεται δυναμικά. Έτσι, οι routers μπορούν να επιλέγουν εναλλακτικούς δρόμους για ένα πακέτο σε περίπτωση που μια συγκεκριμένη σύνδεση του δικτύου παρουσιάζει πρόβλημα και βρίσκεται προσωρινά σε αχρηστία.

- ***TCP/IP stack***

Το TCP/IP stack που χρησιμοποιείται εδώ, έχει αναπτυχθεί ειδικά για την οικογένεια των μικροεπεξεργαστών 8051.Σε αντίθεση με τις πλέον περίτεχνες στοίβες για προσωπικούς υπολογιστές, οι προδιαγραφές του σχετικά με το υλικό μέρος δεν είναι τόσο απαιτητικές. Ένας πλήρης web server μπορεί να εγκατασταθεί κάνοντας χρήση μνήμης RAM λιγότερο από 1KB και περίπου 12 KB κώδικα. Ο μηχανισμός αυτής της στοίβας αποτελεί λογισμικό ανοιχτού πη-
γαίου κώδικα, δηλαδή ο πηγαίος κώδικας είναι γενικώς διαθέσιμος προς κάθε κατεύθυνση. Στην βασική της μορφή, η στοίβα αυτή μπορεί να διαχειριστεί τα περισσότερα από τα πλέον ενδιαφέροντα πρωτόκολλα επικοινωνίας μέσω διαδικτύου όπως το ICMP, το ARP, το PING, το TCP και το UDP. Αυτό που πρέπει να προστεθεί για να έχουμε έναν web server εί-
ναι ARP και το TCP. Με την στοίβα flexGate TCP/IP είναι δυνατός οποιοσδήποτε επιθυμητός αριθμός ταυτόχρονών συνδέσεων. Η στοίβα είναι ενσωματωμένη στον μεταγλωττιστή μc/51. Πρόκειται για ένα πλήρες περιβάλλον ανάπτυξης εφαρμογών της πρότυπης γλώσσας ANSI C

3.3 ETHERNET

Το Ethernet υπάρχει από το 1980. Απαιτεί τη σύνδεση ενός καλωδίου με όλες τις συσκευές. Αν και, αυτό το σχέδιο καλωδίωσης δίνει την εικόνα του τρόπου που το Ethernet επιτρέπει τις επικοινωνίες στο δίκτυο μέσω ενός κοινού μέσου μετάδοσης, η από σταθμό σε σταθμό σύνδεση, έχει σοβαρότατα προβλήματα αξιοπιστίας. Για παράδειγμα, οι εγχώριες επικοινωνίες από κάποιον σταθμό στον κεντρικό, φέρνουν τα δεδομένα σε / από κάθε σταθμό. Το 10BASE-T εισήχθη το 1990 για να αντιμετωπίσει αυτά τα προβλήματα, χρησιμοποιώντας το στρεπτό ζεύγος. Δύο ζεύγη απαιτούνται για κάθε σταθμό: ένα για την εισερχόμενη κυκλοφορία και ένα για την εξερχόμενη. Σε ένα σύστημα 10BASE-T, ένα σφάλμα καλωδίωσης μεταφέρει τις υπηρεσίες δικτύου, μόνο σε έναν σταθμό.

Το 10BASE-T περιλαμβάνει τους διαγνωστικούς δείκτες, που επιτρέπουν τον εύκολο προσδιορισμό σφαλμάτων. Επειδή προσφέρει πλεονεκτήματα αξιοπιστίας και δαπανών, οι περισσότερες νέες εγκαταστάσεις Ethernet χρησιμοποιούν το στρεπτό ζεύγος. Το νεώτερο γρήγορο Ethernet που χρησιμοποιείται από την CobraNet, συνδέεται ακριβώς όπως το 10BASE-T, εκτός από το ότι έχει δέκα φορές το εύρος ζώνης που απαιτεί ένα ελαφρώς υψηλότερου βαθμού καλώδιο, αλλά συναντά και μερικούς περιορισμούς απόστασης, που δε συναντώνται στο 10BASE-T.

3.3.1 ΚΑΛΩΔΙΟ CAT5

Πρόκειται για το αποκαλούμενο UTP στρεπτό ζεύγος. Είναι παρόμοιο με το πανταχού παρόν τηλεφωνικό καλώδιο, αλλά τα ζευγάρια είναι πιο στενά περιπλεγμένα. Το γρήγορο Ethernet χρησιμοποίησε το CAT5 καλώδιο, κάνοντάς το γνωστό ως 100base-tx. Ένα CAT5 καλώδιο περιέχει, συνήθως, τέσσερα στρεπτά ζεύγη καλωδίων δεδομένων, εκ των οποίων δύο χρησιμοποιούνται πραγματικά από το Ethernet: ένα ζευγάρι για τη μετάδοση και ένα για τη λήψη.

Τα νεώτερα καλώδια κατηγορίας 6 ή 7 όπως το Belden MediaTwist 187A, έχουν ακόμα καλύτερη απόδοση από το CAT5 καλώδιο. Αυτοί οι νεώτεροι τύποι καλωδίων, έχουν λιγότερες απώλειες μέσα στο καλώδιο και είναι πιο ανθεκτικά στην παρέμβαση από εξωτερικές πηγές.

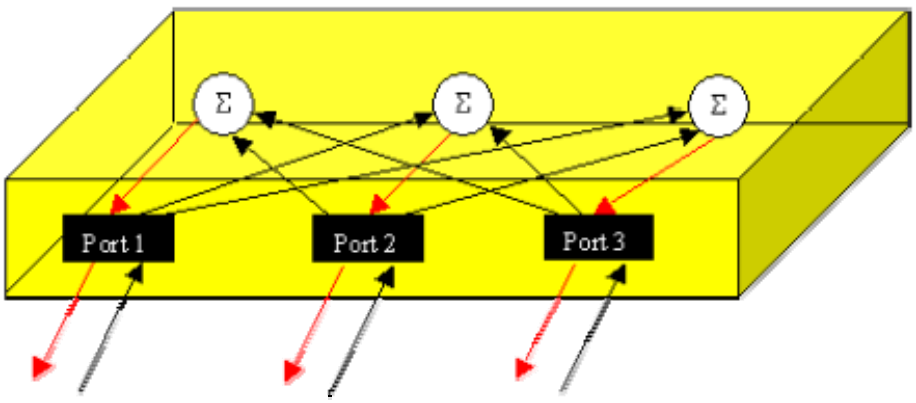
3.3.2 ΚΑΛΩΔΙΑ ΟΠΤΙΚΩΝ ΙΝΩΝ

Υπάρχουν δύο βασικοί τύποι σε χρήση σήμερα: multimode (πολλαπλού τρόπου) και single mode (ενιαίου τρόπου). Η multimode ίνα, χρησιμοποιείται εκτενώς στη βιομηχανία μετάδοσης δεδομένων. Το γρήγορο Ethernet μεταφέρεται μέσω multimode ίνας, γνωστής ως 100base-fx. Το Ethernet μπορεί, μέσω αυτής της ίνας, να μεταφερθεί μέχρι 2 χιλιόμετρα.

Η single mode ίνα, χρησιμοποιείται κυρίως στη βιομηχανία τηλεπικοινωνιών. Αν και δεν υπάρχει κανένα επίσημο πρότυπο για τη μεταφορά Ethernet μέσω ίνας single mode, πολυάριθμα προϊόντα προσφέρουν αυτήν την ικανότητα.

3.3.3 HUB

Γνωστός ως συμπυκνωτής ή επαναλήπτης, αυτή η συσκευή δέχεται τις πολυάριθμες συνδέσεις Ethernet από τις συσκευές δικτύων, τις οποίες και συνδέει χιαστή (cross). Τα δεδομένα που φθάνουν μέσω του ζευγαριού λήψης μιας σύνδεσης, αναπαράγονται και στέλνονται μέσω του ζευγαριού μετάδοσης σε όλες τις συνδεόμενες συσκευές, εκτός από την συσκευή που πραγματοποίησε τη μετάδοση.



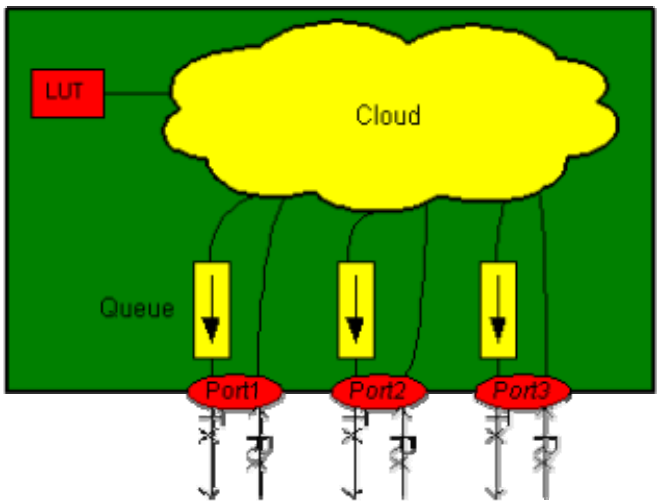
Ένα δίκτυο Ethernet συνδέεται σε συνδεσμολογία αστέρα και το hub είναι στο κέντρο. Τα hubs είναι διαθέσιμα με αριθμήσεις θυρών από 8 έως 24. Μερικά hubs έχουν ικανότητα συσσώρευσης, που επιτρέπει σε πολλαπλά hubs να "δεθούν" μαζί και να ενεργήσουν σαν ήταν μια ενιαία μονάδα, με μεγαλύτερη αρίθμηση θυρών.

Υπάρχουν δύο κατηγορίες γρήγορων hub Ethernet: η κατηγορία 1 και η κατηγορία 2. Τα hubs κατηγορίας 2 έχουν υψηλότερη απόδοση από τα hubs κατηγορίας 1. Τα περισσότερα hubs που χρησιμοποιούνται σήμερα, είναι κατηγορίας 2.

3.3.4 SWITCH

Ένας διακόπτης (switch) είναι μια συσκευή πολλών θυρών, που φιλτράρει και προωθεί πακέτα δεδομένων, μεταξύ των συσκευών δικτύου. Αντίθετα από ένα τυποποιημένο hub, ένας διακόπτης είναι σε θέση να διαβάσει την διεύθυνση προορισμού κάθε πακέτου δεδομένων και να διαβιβάσει, έπειτα, το πακέτο στη σωστή θύρα. Αυτή η "νοημοσύνη" στο διακόπτη, σημαίνει ότι μια δεδομένη συσκευή, λαμβάνει μόνο εκείνα τα πακέτα που απευθύνονται σε αυτήν.

Μια άλλη διαφορά μεταξύ hub και διακόπτη, είναι η έμφυτη δυνατότητα του τελευταίου να αποφεύγει τις συγκρούσεις δεδομένων. Εάν δύο θύρες του διακόπτη προσπαθήσουν να διαβιβάσουν στην ίδια θύρα, τα δεδομένα μπαίνουν σε μια ουρά αναμονής και διαβιβάζονται σειριακά.



Παρόλο που η σύνδεση μεταξύ των διακοπτών έχει ένα όριο εύρους ζώνης, το συνολικό εύρος ζώνης που είναι διαθέσιμο στο δίκτυο, είναι πολύ μεγαλύτερο σε ένα switch δίκτυο. Οι συνδέσεις μεταξύ των διακοπτών και μεταξύ ενός διακόπτη και συσκευών full-duplex, θα λειτουργούν με full-duplex τρόπο. Έτσι, αυξάνεται το όριο από 100 Mbit ανά σύνδεση, σε 100 Mbit ανά κατεύθυνση (σύνολο 200 Mbit).

Με την αποβολή των συγκρούσεων στα δίκτυα switch, λύνεται επίσης και το ζήτημα διαμέτρου του δικτύου. Η εξάλειψη των περιορισμών στη διάμετρο δικτύων, δεν αλλάζει το γεγονός ότι η υποβάθμιση σημάτων λόγω της απόστασης, περιορίζει ένα ενιαίο Cat 5 καλώδιο να εκτείνεται σε 100 μέτρα ή τη multimode ίνα να εκτείνεται σε 2 χιλιόμετρα. Η multimode ίνα μπορεί να χρησιμοποιηθεί στις αποστάσεις μέχρι 100 χιλιόμετρα, ανάλογα με το χρησιμοποιούμενο σύστημα.

Ένα δίκτυο Ethernet που βασίζεται σε διακόπτες, μπορεί να καλωδιωθεί σε διαμόρφωση αστέρα με το διακόπτη στο κέντρο ή σε διαμόρφωση δακτυλίων, χρησιμοποιώντας πολλαπλούς διακόπτες. Όπως και τα hubs, οι διακόπτες είναι

διαθέσιμοι με ποικίλες αριθμήσεις θυρών και πολλοί προσφέρουν την ικανότητα συσσώρευσης. Οι διακόπτες έχουν ικανότητα ανοχής σε σφάλματα, κάτι που δεν παρέχεται με τα hubs.

3.3.5 CROSS-OVER ΚΑΛΩΔΙΟ

Ένα cross-over καλώδιο, μπορεί να χρησιμοποιηθεί για να συνδέσει άμεσα δύο συσκευές, τα hubs ή τους διακόπτες δικτύων. Πρόκειται για ένα CAT5 καλώδιο συνδεδεμένο με τέτοιο τρόπο, ώστε το ζεύγος δεδομένων μετάδοσης από το ένα άκρο, να συνδέεται με το ζεύγος δεδομένων λήψης στο άλλο άκρο.

3.3.6 MEDIA CONVERTER

Ένας τέτοιος μετατροπέας, είναι ουσιαστικά ένα hub δύο θυρών, που δέχεται έναν τύπο μέσου μετάδοσης στη μία θύρα και ένα διαφορετικό τύπο στην άλλη. Οι κοινοί τύποι μέσων Ethernet είναι το στρεπτό ζεύγος και η multimode και single mode ίνα. Μερικά hubs περιλαμβάνουν τη μετατροπή μέσων, μέσω των επιλογών καλωδίωσης μονάδων, για διαφόρους τύπους μέσων.

3.3.7 ΔΙΑΜΕΤΡΟΣ ΔΙΚΤΥΟΥ

Η διάμετρος ενός δικτύου, καθορίζεται από τη μέγιστη απόσταση του καλωδίου μεταξύ δύο οποιωνδήποτε συσκευών του δικτύου και είναι στενά εξαρτώμενη από το χρόνο διάδοσης του δικτύου. Το Ethernet, απαιτεί ότι μια διευκρινισμένη μέγιστη διάμετρος δικτύων δεν ξεπερνιέται σε μια εγκατάσταση, για να εξασφαλίσει ότι οι συγκρούσεις ανιχνεύονται αξιόπιστα, ώστε να επιλύονται. Η διάμετρος δικτύων αποτελεί μεγάλο ζήτημα για τα βασισμένα σε hub δίκτυα, αλλά όχι και για τα switch δίκτυα.

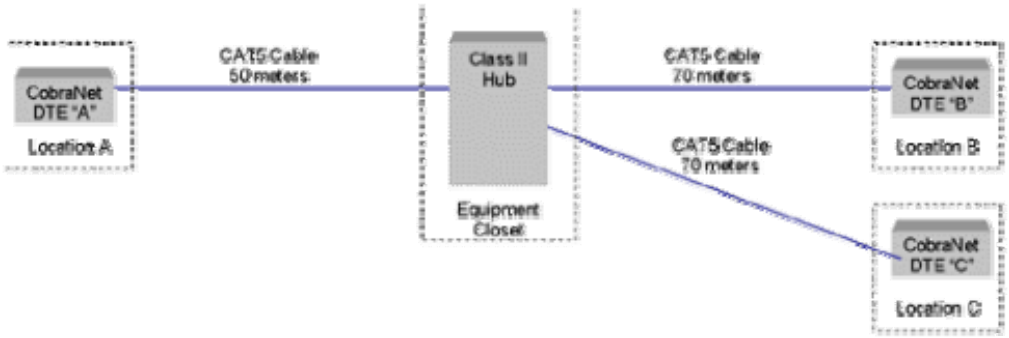
3.3.8 PROPAGATION TIME

Η απόσταση των καλωδίων είναι ένα ζήτημα στα δίκτυα Ethernet, λόγω του χρόνου που απαιτείται για τη διάδοση ενός πακέτου δεδομένων από το ένα άκρο του καλωδίου, στο άλλο. Για να λειτουργεί κατάλληλα το δίκτυο, ένα πακέτο που στέλνεται από οποιαδήποτε συσκευή στο δίκτυο, πρέπει να φθάσει σε όλες τις άλλες συσκευές, μέσα σε ένα ορισμένο χρονικό πλαίσιο. Για να υπολογίσει κανείς το χρόνο μετάδοσης από μια συσκευή του δικτύου σε κάποια άλλη, πρέπει να λογαριάσει όχι μόνο την καθυστέρηση λόγω των καλωδίων, αλλά και την καθυστέρηση των μονάδων (hubs και media converters).

Οι χρόνοι μετάδοσης στα δίκτυα Ethernet, μετριούνται ως κυκλική καθυστέρηση και διευκρινίζονται, για ευκολία, σε μονάδες περιόδου bit. Μια περίοδος bit στο γρήγορο Ethernet είναι 1/100 MHz = 10 nsec. Στον παρακάτω πίνακα, φαίνονται οι χρόνοι μετάδοσης των μονάδων του δικτύου:

Component	**Round Trip Propagation Bit Periods**
Optical Fiber (Single mode and Multimode)	**1.000 / meter**
CAT5 Cable	**1.112 / meter**
Receiver	**>= 100**
Class I Hub	**>= 140**
Class II Hub	**>= 92**

Digi MIL-180 Media converter	**48**
Canary CFT-2132 100BASE-FX / 100BASE-TX Media Converter	**124**
Transition Networks E-100BTX-FRL-01 Media Converter	**133**



Απλό δίκτυο με το hub στο κεντρικό σημείο του εξοπλισμού.

Παρακάτω φαίνεται ένας πίνακας, όπου έχουν υπολογιστεί οι αποστάσεις μετάδοσης μεταξύ των θέσεων Α και Β, του παραπάνω σχήματος.

Component	**Round Trip Propagation Bit Periods**
50 meter Cable between Location A and Hub	**50 x 1.112 = 55.6**
Class II Hub	**92**
70 meter Cable between Hub and Location B	**70 x 1.112 = 77.8**
Location B Receiver	**100**
Total	**325.4**

Οι κυκλικοί χρόνοι μετάδοσης είναι συμμετρικοί: ο χρόνος μετάδοσης από τη θέση Β στη θέση Α και ξανά πίσω στη θέση Β, είναι ο ίδιος με το χρόνο μετάδοσης από τη θέση Α στη θέση Β και ξανά πίσω.

Μελετώντας το διάγραμμα, είναι προφανές ότι ο χρόνος μετάδοσης μεταξύ της θέσης Α και της θέσης Γ, είναι ο ίδιος όπως μεταξύ της θέσης Α και της θέσης Β.

Υπό αυτήν τη μορφή, ο χρόνος μετάδοσης μεταξύ της θέσης Β και της θέσης Γ υπολογίζεται:

Component	Round Trip Propagation Bit Periods
70 meter Cable between Location B and Hub	**70 x 1.112 = 77.8**
Class II Hub	**92**
70 meter Cable between Hub and Location C	**70 x 1.112 = 77.8**
Location C Receiver	**100**
Total	**347.6**

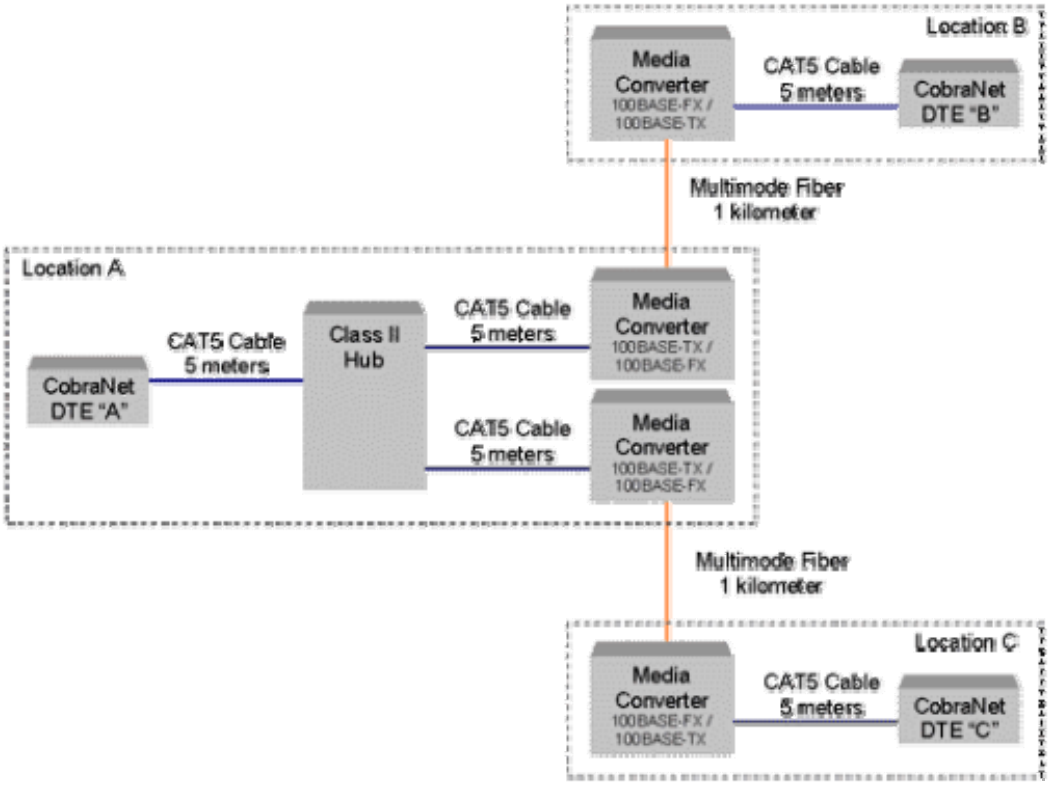
Ο χρόνος μετάδοσης των 347,6 bit μεταξύ της θέσης Β και της θέσης Γ, τον καθιστά το μεγαλύτερο στο δίκτυο. Μπορούμε να πούμε δηλαδή, ότι αυτό το δίκτυο έχει μια διάμετρο των 347,6 περιόδων bit.

3.3.9_ ΟΡΙΟ ΔΙΑΜΕΤΡΟΥ

Τα πρότυπα του Ethernet, απαιτούν μέγιστη διάμετρο δικτύου της τάξεως των 512 περιόδων bit. Επιτρέποντας χρόνους μετάδοσης μέσω μέχρι δύο hubs και του δέκτη, το γρήγορο Ethernet υποστηρίζει μια μέγιστη διάμετρο μόλις πάνω από 200 μέτρα, σε ένα hub δίκτυο.

Με την επέκταση μερικών παραμέτρων συγχρονισμού, η CobraNet που χρησιμοποιεί hub δίκτυα, μπορεί να ανεχτεί διαμέτρους δικτύων μέχρι 2560 περιόδων bit. Επιτρέποντας χρόνους μετάδοσης μέσω μέχρι δύο hubs και του δέκτη, υποστηρίζεται μια μέγιστη διάμετρος άνω των 2 χιλιομέτρων.

Παρακάτω φαίνεται ένα σύστημα διανομής, με την κεντρική θέση ελέγχου (θέση Α) και δύο δορυφορικές θέσεις (Β και Γ), που συνδέονται μέσω ίνας.



Ο παρακάτω πίνακας τέλος, απεικονίζει τους χρόνους μετάδοσης από τη θέση Β στη θέση Γ, μέσω της θέσης Α.

Component	Propagation Bit Periods
5-meter CAT5 cable between CobraNet "B" and Media Converter	5 x 1.112 = 5.6
100BASE-TX / 100BASE-FX media conversion at Location B	48
1K-meter fiber between Location B and Location A	1000 x 1.0 = 1000
100BASE-FX / 100BASE-TX media conversion at Location A	48
5-meter CAT5 cable between Media Converter and Hub	5 x 1.112 = 5.6
Class II Hub	92
5-meter CAT5 cable between Hub and Media Converter	5 x 1.112 = 5.6
100BASE-TX / 100BASE-FX media conversion at Location A	48
1K-meter fiber between Location A and Location C	1000 x 1.0 = 1000
100BASE-FX / 100BASE-TX media conversion at Location C	48
5-meter CAT5 cable between Media Converter and CobraNet device "C"	5 x 1.112 = 5.6
CobraNet device "C" Receiver	100
Total	2406.4
Maximum CobraNet Diameter	2560
Margin	153.6

3.3.10_ΓΡΗΓΟΡΟ ETHERNET

Το γρήγορο Ethernet (επίσης γνωστό ως 100 Mbps Ethernet), υποστηρίζει ένα μέγιστο data rate 100 Mbps. Ονομάζεται έτσι, επειδή η αρχική τεχνολογία Ethernet υποστήριξε μόνο 10 Mbps. Το γρήγορο Ethernet άρχισε να επεκτείνεται ευρέως στα μέσα της δεκαετίας του '90, καθώς η ανάγκη για μεγαλύτερη απόδοση του LAN, έγινε κρίσιμη στα πανεπιστήμια και τις επιχειρήσεις.

Ένα βασικό στοιχείο της επιτυχίας του γρήγορου Ethernet, ήταν η δυνατότητά του να συνυπάρχει με τις υπάρχουσες εγκαταστάσεις δικτύων. Σήμερα, πολλοί προσαρμογείς (adapters) δικτύων, υποστηρίζουν τόσο το παραδοσιακό, όσο και το γρήγορο Ethernet. Αυτοί οι αποκαλούμενοι "10/100" adapters, μπορούν συνήθως να αισθανθούν την ταχύτητα της γραμμής αυτόματα και να ρυθμίσουν ανάλογα τις παραμέτρους.

3.3.11 GIGABIT ETHERNET

Ακριβώς όπως το γρήγορο Ethernet είναι βελτιωμένο σε σχέση με το παραδοσιακό Ethernet, το Gigabit Ethernet είναι βελτιωμένο σε σχέση με το γρήγορο Ethernet, προσφέροντας data rate 1000 Mbps, αντί 100 Mbps. Το Gigabit Ethernet σχεδιάστηκε αρχικά για να μεταδίδεται μέσω καλωδίου χαλκού ή οπτικών ινών, αλλά τα πρότυπα 1000Base-T, επίσης το υποστηρίζουν επιτυχώς. Το 1000Base-T

χρησιμοποιεί την καλωδίωση κατηγορίας 5, παρόμοια με του 100 Mbps Ethernet, αν και η τάξεως gigabit ταχύτητα, απαιτεί την χρήση επιπρόσθετων ζευγών καλωδίων.

3.3.12 ΤΟ ΧΡΗΣΙΜΟΠΟΙΟΥΜΕΝΟ ETHERNET

Ένα δίκτυο Ethernet λειτουργεί σχεδόν ιδανικά, ασχέτως ταχύτητας και σχεδιαγράμματος. Οι συσκευές που συνδέονται με το δίκτυο, κατέχουν μια χαρακτηριστική interface κάρτα δικτύου (NIC -**N**etwork **I**nterface **C**ard-) -γενικότερα ένας προσαρμογές δικτύου-, που διασυνδέεται άμεσα στο bus. Το NIC περιλαμβάνει έναν connector καλωδίων όπως ο RJ-45 connector, που χρησιμοποιείται με τα σύγχρονα τηλέφωνα (οι αρχικές εκδόσεις Ethernet, παρόλα ταύτα, χρησιμοποιούσαν πολύ διαφορετικούς connectors).

Τα δεδομένα που στέλνονται πέρα από το Ethernet, υπάρχουν σε μορφή frames. Ένα frame Ethernet, περιέχει δύο επιγραφές και ένα τμήμα δεδομένων, που έχει ένα συνδυασμένο μήκος λιγότερο από 1518 byte. Τα πρότυπα απαιτούν τα frames να είναι ραδιοφωνικά συζευγμένα με όλες τις συσκευές, που σημαίνει ότι οι προσαρμογείς δικτύων, πρέπει ρητά να αναγνωρίσουν και να απορρίψουν όλα τα frames, που δεν ήταν προορισμένα να παραληφθούν από αυτούς.

Οι συσκευές που θέλουν να διαβιβάσουν μέσω Ethernet, εκτελούν αρχικά έναν πολύ γρήγορο έλεγχο, για να καθορίσουν εάν το μέσο είναι διαθέσιμο ή εάν μια μετάδοση βρίσκεται σε εξέλιξη τη δεδομένη χρονική στιγμή. Εάν το Ethernet είναι διαθέσιμο, η συσκευή διαβιβάζει. Εντούτοις, τα πρότυπα Ethernet δεν αποτρέπουν τις πολλαπλές συσκευές από το να διαβιβάσουν ακαριαία. Αυτές οι συγκρούσεις έχουν ως συνέπεια, από τη μία την αποτυχία της μετάδοσης και από την άλλη και οι δύο συσκευές να αναμεταδώσουν σε ένα μεταγενέστερο χρονικό διάστημα. Ένας εξειδικευμένος αλγόριθμος χρησιμοποιείται με το Ethernet, για να καθορίσει τον κατάλληλο χρόνο αναμονής μεταξύ των αναμεταδόσεων.

Όπως αναφέρεται ανωτέρω, τόσο τα καλώδια Ethernet είναι περιορισμένα στην προσιτότητά τους, αλλά και οι αποστάσεις (της τάξεως των 100 μέτρων), είναι ανεπαρκείς να καλύψουν τις μεσαίου και μεγάλου μεγέθους, ομάδες εργασίας. Ένας επαναλήπτης (repeater) στο networking του Ethernet, είναι μια συσκευή που επιτρέπει στα πολλαπλά καλώδια, να είναι ενωμένα και να εκτείνονται σε μεγαλύτερες αποστάσεις.

3.3.13 MODBUS / TCP

Η διαδεδομένη χρήση του TCP/IP προσφέρει μια σημαντική προοπτική: οι slave συσκευές, να μπορούν να υποστηριχτούν από όλα τα σημαντικά πρωτόκολλα. Φαινομενικά χαοτικό, κάτι τέτοιο θα μπορούσε να επαναπροσδιοριστεί κατά τη διάρκεια του χρόνου, ώστε να δημιουργήσει ένα ενιαίο βιομηχανικό πρωτόκολλο.

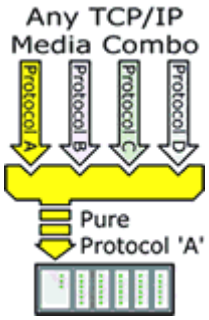
Το αξιόπιστο, κατά το συντάκτη, "πάντρεμα" του Ethernet και του TCP/IP με πολλά πρωτόκολλα σε ένα ενιαίο καλώδιο και σε μια ενιαία συσκευή, προσδίδει έναν εξελικτικό χαρακτήρα στον αυτοματισμό.

Ένας από τους πολλούς περιορισμούς των σειριακών μεταδόσεων δεδομένων, είναι η απουσία οποιουδήποτε 100% αξιόπιστου τρόπου ανάμιξης των πολλαπλών πρωτοκόλλων, σε ένα ενιαίο φυσικό καλώδιο. Λεπτές διαφορές μεταξύ του framing των πακέτων, του χρονισμού και της ειδικής επεξεργασίας της ακολουθίας

χαρακτήρων, απαιτούνται από κάθε πρωτόκολλο για να “επιβιώσει” στο ίδιο του το καλώδιο. Αυτός ο περιορισμός, είναι μια πρωταρχική αιτία του πονοκεφάλου των εξελισσομένων, κατά τη διάρκεια του χρόνου, multi-vendor συστημάτων (αν και ακόμα και επιτυχημένες γενεές single-vendor συστημάτων προσφέρουν, συχνά, αυτόν τον πονοκέφαλο).

Στο TCP/IP εκτός από την απανταχού παρούσα διεύθυνση IP, κάθε πακέτο δεδομένων περιλαμβάνει έναν αριθμό θυρών, από το 1 έως το 65535. Εάν μια διεύθυνση IP είναι (όπως και στον ενιαίο κύριο αριθμό τηλεφώνου) ο καθοδηγητής της κλήσης σε κάποιο συγκεκριμένο σημείο, ο αριθμός θυρών είναι ο αριθμός επέκτασης. Δεδομένου ότι το TCP/IP επιτρέπει πάρα πολλές εφαρμογές να εκδηλώσουν ενδιαφέρον για τα δεδομένα ενός ενιαίου καλωδίου, είναι πραγματικά τετριμμένο για τα πολλαπλά πρωτόκολλα, να μοιραστούν το ίδιο καλώδιο. Όντας αυτονόητο ότι κάθε πακέτο στοιχείων λαμβάνεται από κάπου, το TCP/IP το καθοδηγεί στη σωστή εφαρμογή. Επιπλέον, κάθε εφαρμογή μπορεί να αναμείνει ένα διαφορετικό πρωτόκολλο, έτσι ώστε να μην υπάρχει κανένας λόγος μια ενιαία συσκευή, να μην μπορεί να υποστηρίξει συγχρόνως, την πληθώρα των πρωτοκόλλων.

Παραδείγματος χάριν, ένας υπολογιστής ροής με ένα ιδιόκτητο πρωτόκολλο ενσωματωμένο στο TCP/IP (πρωτόκολλο "A" όπως ονομάστηκε στο επόμενο σχήμα), μπορεί να αξιοποιήσει πλήρως αυτό το πρωτόκολλο για τη διαμόρφωση και τη συντήρηση. Αυτό δεν εμποδίζει την προσθήκη κώδικα, ώστε να επιτρέψει στο Modbus/TCP, το Profibus και τα πρωτόκολλα CIP κάτω από το TCP/IP, να διαβάζουν/γράφουν τα real time δεδομένα του υπολογιστή ροής. Επιπλέον, αντίθετα από τις παραδοσιακές σειριακές επικοινωνίες, το TCP/IP επιτρέπει την multi-access (πολλαπλή πρόσβαση) και δεν έχει κανένα έμφυτο όριο, που να περιορίζει τους χρήστες σε έναν ενιαίο host. Τέλος, είναι εύκολο να εφαρμοστεί ένα master σύστημα, δεδομένου ότι κάθε master μπορεί να έχει ανεξάρτητη πρόσβαση στον κάθε slave.



3.3.13 ΤΟ ETHERNET ΚΑΙ ΤΑ ΣΤΡΩΜΑΤΑ ΤΗΣ OSI

Δε θα αναλύσουμε λεπτομερώς τη λειτουργία των διαφόρων στρωμάτων στο πρότυπο της OSI. Θα εστιάσουμε καθαρά στο Ethernet και, επίσης, στο κυρίαρχο πρωτόκολλο που τρέχει στο Ethernet: το TCP/IP.

- ***ΣΤΡΩΜΑΤΑ 1 ΚΑΙ 2***

Αρχικά εξετάζουμε το φυσικό επίπεδο. Τα αρχικό μέσο που διευκρινίστηκε, ήταν ένα παχύ ομοαξονικό καλώδιο, γνωστό ως 10base5 στις σχετικές IEEE

προδιαγραφές. Τα μειονεκτήματα σχετικά με το κόστος και την ευκολία της εγκατάστασης, οδήγησαν στη εισαγωγή του 10base2 ή Thin Ethernet. Αργότερα προέκυψαν περισσότερες επιλογές: ευρεία ζώνη, διάφοροι τύποι ινών και unshielded στρεπτά ζεύγη.

Τα βασικά σημεία της διευθυνσιοδότησης του Ethernet είναι:

• Σε κάθε συσκευή Ethernet ορίζεται μια μοναδική διεύθυνση των 48 bit.
• Οι τρόποι διευθυνσιοδότησης είναι οι single point (ενιαίου σημείου), multi-cast (πολλαπλής διανομής) ή broadcast (ραδιοφωνική μετάδοση).
• Οι κόμβοι θα αγνοήσουν τα μηνύματα που δεν είναι για αυτούς.
• Οι συσκευές έξυπνων δικτύων θα μεταφέρουν το φορτίο στον κύριο μικροελεγκτή.

Τα βασικά σημεία της τοπολογίας του Ethernet είναι:

• Αρχικά η τοπολογία ήταν multi-drop.
• Για να βελτιωθεί η απόδοση σε βαριά φορτία, εισήχθησαν οι συσκευές.
• Τα multi-port hubs έχουν διευκολύνει τις χρησιμοποιούμενες τοπολογίες αστέρα.
• Οι γέφυρες επιτρέπουν τη χρήση των τμημάτων Ethernet, χρησιμοποιώντας διαφορετικά φυσικά μέσα.
• Μια σχετικά νέα ανάπτυξη μεταστρέφει τα hubs ή τους διακόπτες στρώματος 2.
• Τα μεταστρεπτόμμενα hubs, δίνουν πολύ μεγαλύτερο data rate από τα επαναληπτικά hubs.

• *ΤΑ ΘΕΜΕΛΙΩΔΗ ΠΡΟΒΛΗΜΑΤΑ ΕΦΑΡΜΟΓΗΣ ΤΟΥ ETHERNET ΣΤΗ ΒΙΟΜΗΧΑΝΙΑ*

Θυμίζουμε, αρχικά, ότι εξετάζουμε μόνο τα στρώματα 1 και 2. Το θεμελιώδες πρόβλημα με το μηχανισμό CSMA/CD που περιγράφεται στο αρχικό IEEE 802.3, είναι ότι χρησιμοποιεί την ανίχνευση σύγκρουσης και υπαναχωρεί, έτσι ώστε το δίκτυο να μοιραστεί. Αυτή η μέθοδος, όμως, είναι πλήρως μη αιτιοκρατική. Εάν αυτό το καθιστά ακατάλληλο προς χρήση σε βιομηχανικό περιβάλλον, εξαρτάται αρχικά από την εφαρμογή και επίσης από την ταχύτητα του δικτύου.

Το θεμελιώδες χαρακτηριστικό των δικτύων Ethernet σε οποιαδήποτε ταχύτητα, είναι ότι η μέση ταχύτητα υποβιβάζεται με την υπερφόρτωση, με ένα μη γραμμικό τρόπο.

Υπάρχει μια επέκταση στα πρότυπα, με σκοπό να εξετάσει αυτό το ιδιαίτερο πρόβλημα (IEEE 802.1p) και επιτρέπει στους σχεδιαστές συστημάτων να δώσουν προτεραιότητα στα μηνύματα, ώστε να εγγυηθούν την παράδοση των κρίσιμων χρονικά δεδομένων, δίνοντας κατά συνέπεια τους αιτιοκρατικούς χρόνους απόκρισης. Επιπλέον, μερικοί προμηθευτές ελέγχου έχουν αναπτύξει λύσεις, χρησιμοποιώντας τον ιδιόκτητο κώδικα στον τυποποιημένο σωρό TCP/IP. Ένα παράδειγμα αυτής της προσέγγισης είναι εφαρμογή Hewlett Packard που επιτρέπει το συγχρονισμό των κόμβων, σε περίπου 200 nsec.

Μια άλλη επέκταση με σκοπό τη βελτίωση της απόδοσης, είναι το IEEE802.3x. Αυτό επιτρέπει την αμφίδρομη ταυτόχρονη μετάδοση και την υποδοχή των τυποποιημένων frames Ethernet, με μέσα ικανά να υποστηρίξουν αυτό το χαρακτηριστικό γνώρισμα. Αυτό αναφέρεται ως full-duplex (πλήρως διπλός) τρόπος.

Άλλο θεμελιώδες πρόβλημα με το πρότυπο Ethernet, είναι ο τρόπος παροχής στον πλεονασμό. Πολλές ιδιόκτητες λύσεις έχουν προκύψει στον εμπορικό κόσμο, αλλά εάν εξετάζουν πλήρως τα βιομηχανικά προβλήματα, εξαρτάται και πάλι από την εφαρμογή. Μια επέκταση που μπορεί να αποβάλει την ανάγκη για τις ιδιόκτητες λύσεις, είναι η IEEE802.12. Αυτή παρέχει τη δυνατότητα να προστεθούν οι περιττές συνδέσεις με το δίκτυο, για να διευκολυνθεί η αυτόματη αποκατάσταση της

συνδετικότητας του δικτύου, όταν υπάρχει μια αποτυχία σύνδεσης ή επαναληπτών, σε οποιοδήποτε path του δικτύου. Αναμφισβήτητα υπάρχουν περαιτέρω εκτιμήσεις για τις πιο επίπονες απαιτήσεις (π.χ. συστήματα ασφάλειας).

- ***ΣΤΡΩΜΑ 3***

Η ανάπτυξη του Internet και η υιοθέτηση του πρωτοκόλλου του (IP), έχουν καθιερώσει το IP ως τον πυρήνα του παγκοσμίου επιχειρηματικού δικτύου σε όλα τα επίπεδα (Internet και Intranet).

Ξαναγυρίζοντας στο πρότυπο επτά στρωμάτων, το χαμηλότερο στρώμα του TCP/IP είναι το στρώμα network, όπου βρίσκεται και το IP. Ο λόγος που το IP μπορεί να κινήσει τα δεδομένα από τα εταιρικά Intranets προς το παγκόσμιο Internet, είναι το ευρύ φάσμα των τεχνολογιών του LAN ή WAN που υποστηρίζονται.

Στο στρώμα network, βρίσκεται επίσης και το πρωτόκολλο arp. Το arp χρησιμοποιείται για να χαρτογραφήσει τις διευθύνσεις Ethernet στις διευθύνσεις IP και για να διατηρήσει τους πίνακες χαρτογράφησης, σε κάθε συσκευή στο δίκτυο. Οι διευθύνσεις IP είναι μοναδικές σε οποιοδήποτε δίκτυο, αλλά αντίθετα από τις διευθύνσεις Ethernet που καθορίζονται από το υλικό του κόμβου, είναι διαμορφώσιμες από τον χρήστη. Οποιαδήποτε αλλαγή, εντούτοις, πρέπει να γίνει με έναν ελεγχόμενο και προγραμματισμένο τρόπο, λαμβάνοντας υπόψη όλες τις πιθανές συνέπειες.

Όπως και στις διευθύνσεις Ethernet, οι διευθύνσεις IP μπορούν να είναι unicast, multicast ή broadcast. Η τεχνολογία IP και οι δυνάμεις αγοράς πίσω από αυτό, έχουν παρουσιάσει τεράστια δυνατότητα να προσαρμόσουν και να χρησιμοποιήσουν τις νέες τεχνικές δικτύωσης και τις τεχνολογίες επικοινωνιών, όπως αυτές προκύπτουν. Το Internet είναι η σταθερή απόδειξη των δικτύων IP, τα οποία είναι ικανά εξελιξιμότητας. Επίσης οι λύσεις ασφαλείας, εμφανίζονται τώρα που καλύπτουν ένα ευρύ φάσμα των αναγκών (κρυπτογράφηση, επικύρωση, ψηφιακές υπογραφές κ.λ.π.).

- ***ΣΤΡΩΜΑ 4***

Οι μεταφορές που υποστηρίζονται από την ακολουθία πρωτοκόλλου TCP/IP, είναι το TCP (πρωτόκολλο ελέγχου μετάδοσης) και UDP (πρωτόκολλο στοιχείων χρηστών). Το TCP είναι μια προσανατολισμένη προς τη σύνδεση μεταφορά, που παρέχει την αξιόπιστη διαβίβαση δεδομένων από μια συσκευή σε κάποια άλλη. Παρουσιάζει τα δεδομένα στο στρώμα application από επάνω του, υπό μορφή συνεχής ροής byte. Το UDP είναι ένα πολύ απλούστερο πρωτόκολλο μεταφορών. Χρησιμοποιείται από τις εφαρμογές που εφαρμόζουν τη χειραψία (handshaking) μεταξύ δύο συσκευών και απαιτούν μόνο μια ελάχιστη υπηρεσία μεταφορών.

Αρχικά αναπτυγμένο για το ARPANET υπό τη χρηματοδότηση του Υπουργείου Αμύνης των ΗΠΑ, το TCP/IP χρησιμοποιείται στις περισσότερες πλατφόρμες υπολογιστών και ενσωματώνεται σε κάθε αντίγραφο των Windows και των λειτουργικών συστημάτων των Windows NT.

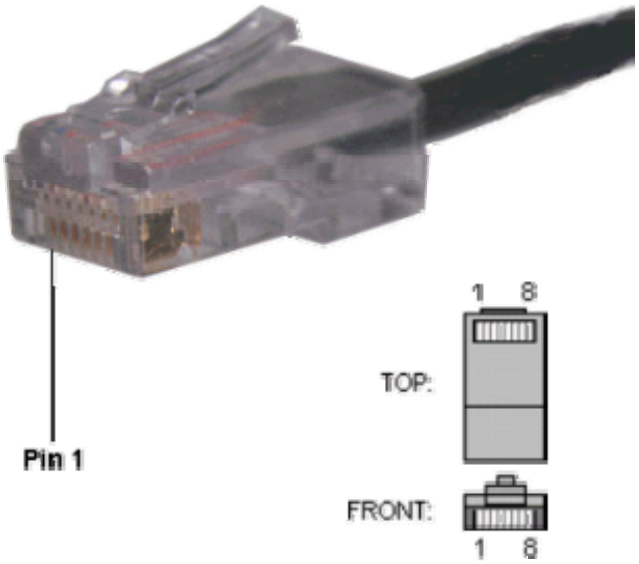
Ένα σημαντικό όφελος του Ethernet είναι ότι μέσω του TCP/IP, είναι το κύριο δίκτυο τόσο για το Intranet (εμπορικό δίκτυο μέσα στο εργοστάσιο), όσο και για το Internet (για τη σύνδεση μεταξύ των εργοστασίων).

- ***ΣΤΡΩΜΑΤΑ 5, 6 ΚΑΙ 7***

Οι υπηρεσίες χρηστών που συνδέονται συνήθως με τα δίκτυα TCP/IP, χαρτογραφούνται άμεσα στο στρώμα 7. Το TCP/IP δεν έχει καμία συγκεκριμένη χαρτογράφηση στα στρώματα 5 και 6 του προτύπου και δεν εγγυάται από μόνο του, ότι δύο συσκευές μπορούν να επικοινωνήσουν η μία με την άλλη. Πρέπει να υπάρχει συμβατότητα με τα στρώματα εφαρμογής.

- ***RJ-45***

Ο RJ 45 χρησιμοποιείται συνήθως για δίκτυο και για εφαρμογές τηλεφωνίας . Χρησιμοποιείται επίσης για τις σειριακές συνδέσεις σε ειδικές περιπτώσεις.



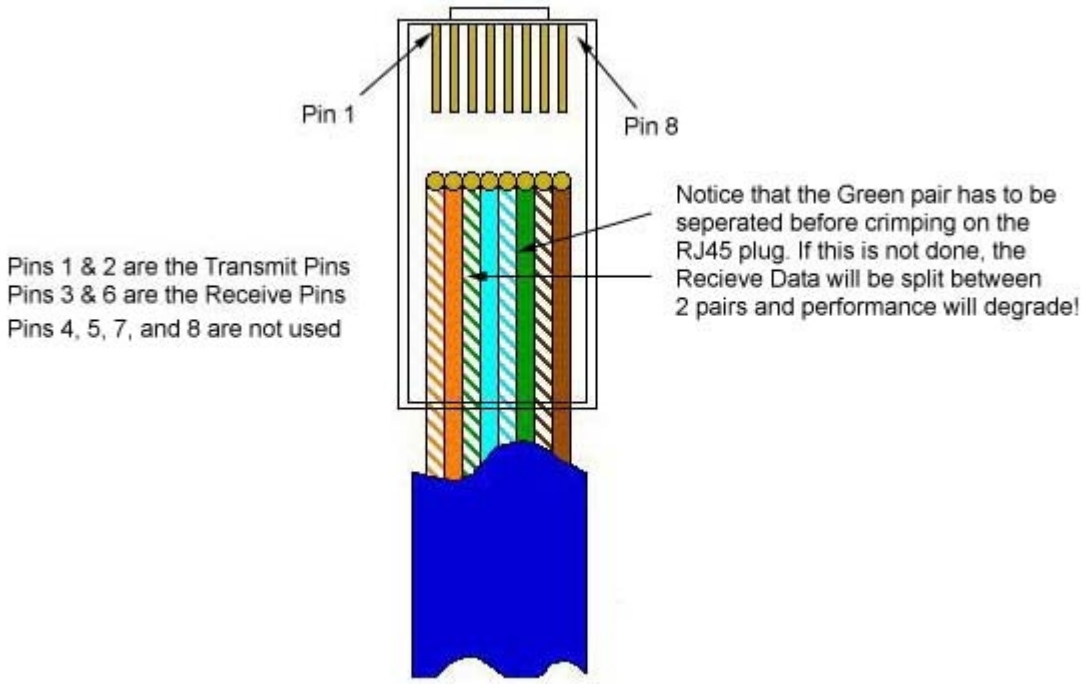
- ***Pinout για Ethernet***

Αν και χρησιμοποιείται για ποικίλους λόγους, RJ 45 κυρίως χρησιμοποιείται για τις συνδέσεις 10 Base-T και 100 Base-TX Ethernet συνδέσεις.

Pin #	Ethernet 10BASE-T 100BASE-TX	EIA/TIA 568A	EIA/TIA 568B or AT&T 258A
1	Transmit +	White with green strip	White with orange stripe
2	Transmit -	Green with white stripe or solid green	Orange with white stripe or solid orange
3	Receive +	White with orange stripe	White with green stripe
4	N/A	Blue with white stripe or solid blue	Blue with white stripe or solid blue
5	N/A	White with blue stripe	White with blue stripe
6	Receive -	Orange with white stripe or solid orange	Green with white stripe or solid
7	N/A	White with brown strip or solid brown	White with brown strip or solid brown
8	N/A	Brown with white stripe or solid brown.	Brown with white stripe or solid brown.

Επειδή μόνο δύο ζευγάρια των καλωδίων στον RJ 45 χρησιμοποιούνται για να φέρνουν σήματα Ethernet, και οι δύο 10BASE-T και 100BASE-TX χρησιμοποιούν τα ίδια καλώδια πάνω στο βύσμα. ένα καλώδιο διασταυρώσεων δημιουργήθηκε για το ένα και εργάζεται και για το άλλο

Επίσης, παρακαλώ σημειώστε ότι είναι πολύ σημαντικό ότι ένα ενιαίο ζευγάρι χρησιμοποιείται για τα καλώδια 3 και 6. Εάν ένας αγωγός από ένα ζευγάρι χρησιμοποιείται για το καλώδιο 3 και ένας αγωγός από ένα άλλο ζευγάρι χρησιμοποιείται για το καλώδιο 6, η απόδοση θα υποβιβάσει. Δείτε το ακόλουθο σχήμα.



- ***RJ-45 Pinout για RocketPort***

Το ακόλουθο διάγραμμα παρουσιάζει το pinout του RJ45 χρησιμοποιούμενο σε ορισμένες κάρτες σειριακών επαφών RocketPort .

Pin	Name/Description
1	Request To Send
2	Data Terminal Ready
3	Ground
4	Transmit Data
5	Receive Data
6	Data Carrier Detect
7	Data Set Ready
8	Clear To Send

- ***Pinouts για ISDN***
  - Εδώ είναι ένας ISDN BRI pinout για ένα Cisco 750 σειρών:

Το ακόλουθο διάγραμμα παρουσιάζει το pinout του RJ45 χρησιμοποιούμενο σε ένα ορισμένο ISDN S/T .

Pin	Color	Name/Description
1	White/Orange	N/A
2	Orange	N/A
3	White/Green	Receive+
4	Blue	Transmit +
5	White/Blue	Transmit -
6	Green	Receive -
7	White/Brown	-48VDC (optional)
8	Brown	-48VDC Return (optional)

Pin	Function
1	Not used
2	Not used
3	Not used
4	U interface network connection (tip)
5	U interface network connection (ring)
6	Not used
7	Power (pass-through to S connector)
8	Ground (pass-through to S connector)

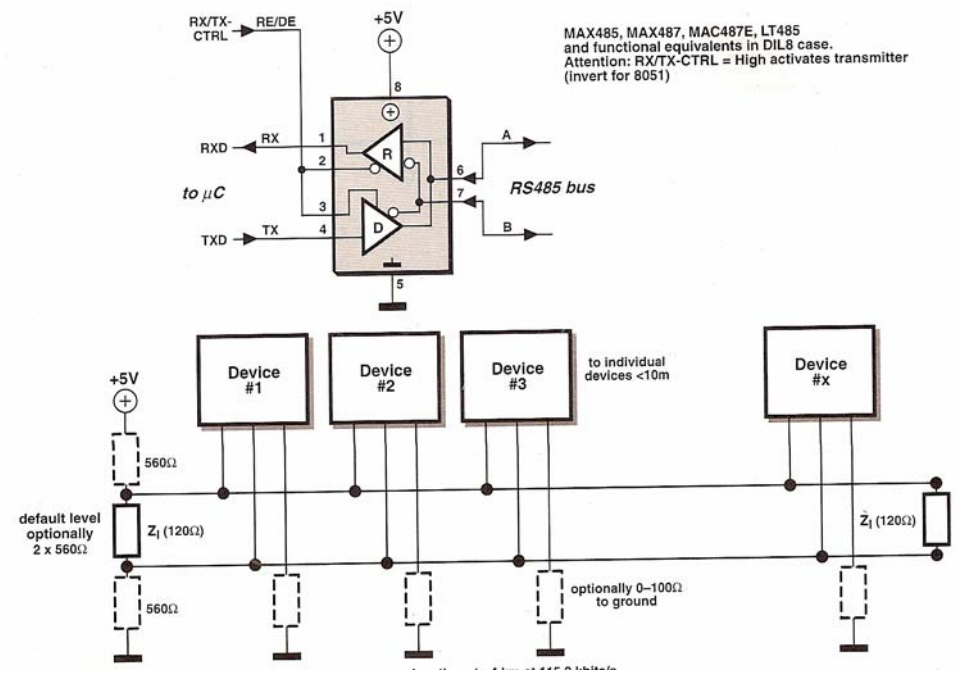
4. ΔΙΚΤΥΟ ΜΕΣΩ RS 485

Η πλακέτα του μικροελεγκτή MSC 1210 είναι εξοπλισμένη με μια θύρα επικοινωνιών RS485. Με την βοήθεια της συνδέονται μεταξύ τους πολλές πλακέτες ανταλλάσσοντας δεδομένα με υψηλούς ρυθμούς μετάδοσης. Η έκταση του σχηματιζόμενου δικτύου ξεπερνάει το μήκος το ένα χιλιόμετρο.

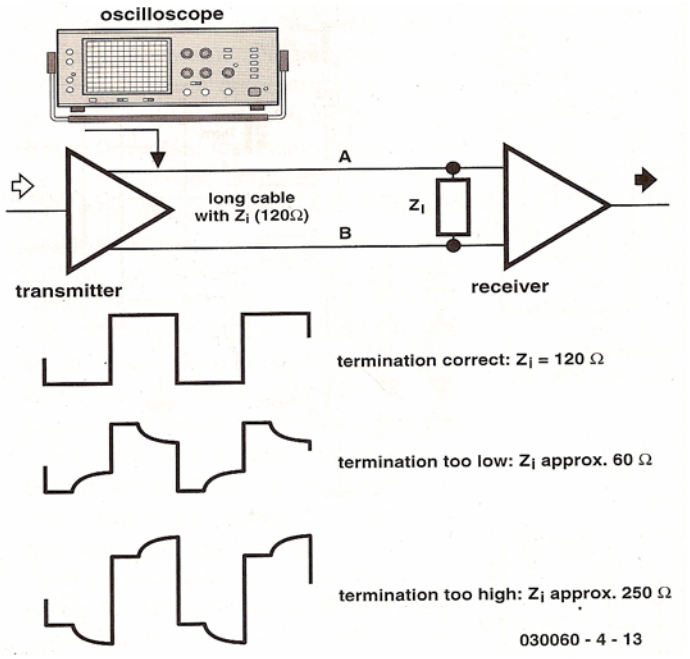
Αν και η διασύνδεση RS485 είναι γνωστή από τις πρώτες μέρες των προσωπικών υπολογιστών εδώ και 20 χρόνια την συναντάμε σπάνια στα σημερινά μηχανήματα. Αυτό όμως με κανένα τρόπο δεν σημαίνει πως οι σύγχρονοι PC αδυνατούν να τα βγάλουν πέρα με ένα τέτοιο κύκλωμα. Το απλούστερο που θα μπορούσε να σκεφτεί κάποιος προκειμένου να εξοπλίσει τον υπολογιστή του με μια θύρα RS485 θα ήταν η χρήση μιας εξειδικευμένης θυγατρικής πλακέτας τοποθετημένη σε μια ελεύθερη υποδοχή PCI.

4.1 Τοπολογία διαύλου:

Ένα δίκτυο αποτελείται από ένα ζευγάρι αγωγών μεγάλου μήκους στα άκρα των οποίων έχουν αντιστάσεις τερματισμού. Αν οι διασυνδεόμενες συσκευές χρησιμοποιούν συνηθισμένες μεθόδους προσπέλασης, το πλήθος των μονάδων που μπορούν συνδεθούν στους αγωγούς φθάνει τις 32. Αν όμως, χρησιμοποιηθούν εξειδικευμένα ολοκληρωμένα κυκλώματα οδήγησης τότε το πλήθος των μονάδων αυξάνεται στις 256. Θεωρητικά όλες οι συσκευές συνδέονται πάνω στο ζεύγος των αγωγών του διαύλου με καλώδια μικρού μήκους. Στην πράξη όμως το μήκος των καλωδίων σύνδεσης είναι πολύ μεγαλύτερο σχηματίζοντας εμφανείς διακλαδώσεις όπως φαίνεται στο πιο κάτω σχήμα



Στην απλούστερη περίπτωση το δίκτυο RS485 υλοποιείται από δύο μόνο αγωγούς , τους Α και Β, οι οποίοι συνοδεύονται από τις κατάλληλες αντιστάσεις τερματισμού. Αν και αυτή η τοπολογία φαίνεται να πληρεί όλες τις προδιαγραφές του διαύλου πολλές φορές χρησιμοποιούνται επιπλέον αντιστάσεις συνδεμένες σε ένα τρίτο αγωγό. Μια άλλη εξίσου δημοφιλής παραλλαγή του διαύλου RS485 απαιτεί την παρουσία τεσσάρων αγωγών. Οι δύο από αυτούς αξιοποιούνται για την μεταφορά δεδομένων, ενώ οι πλεονάζοντες για την τροφοδότηση των συσκευών που ταξιδεύουν στους αγωγούς μεταφέρουν την πληροφορία τροποποιώντας τις ηλεκτρικές στάθμες ηρεμίας. Αν η διαφορά των τάσεων στους δύο αγωγούς είναι θετική τότε λέμε πως ο δίαυλος μεταφέρει λογικό ‘1’. Αν είναι αρνητική τότε ο δίαυλος μεταφέρει λογικό ‘0’. Η χρήση διαφορικών τάσεων κάνει πιο αναίσθητο στον ηλεκτρικό θόρυβο, εφόσον βέβαια πληρούνται και κάποιες επιπρόσθετες προυποθέσεις . Στο πιο κάτω σχήμα φαίνεται ένας δίαυλος στα άκρα του οποίου έχουν συνδεθεί δύο συσκευές. Το σήμα μεταφέρεται από αριστερά προς τα δεξιά με την βοήθεια ενός καλωδίου μεγάλου μήκους. Αν συνδέσουμε έναν παλμογράφο στο σημείο που υποδεικνύει το σχήμα, θα πάρουμε τις κυματομορφές που φαίνονται ακριβώς κάτω από αυτό.

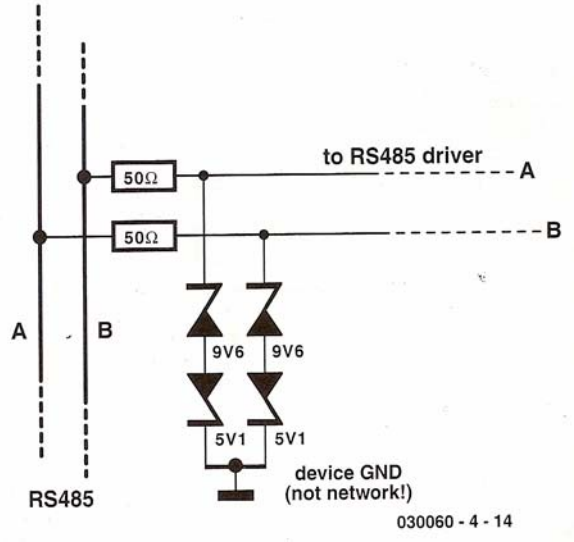


Το ποια απ' όλες θα είναι εκείνη που θα παρατηρήσουμε στο δικό μας όργανο μέτρησης θα εξαρτηθεί από τις τιμές των αντιστάσεων τερματισμού. Οι αντιστάσεις τερματισμού μπορούν να κυμαίνονται από 60 έως 200 Ω. Κριτήριο για το ποια τιμή θα χρησιμοποιήσουμε αποτελεί η σύνθετη αντίσταση του καλωδίου. Είναι προφανές, πως οι αντιστάσεις τερματικού πρέπει να έχουν ίδια τιμή με την ονομαστική αντίσταση του καλωδίου. Η πλέον συνηθισμένη είναι αυτή των 120 Ω. Σημειώνουμε πως λανθασμένη επιλογή των αντιστάσεων τερματισμού προκαλεί παραμορφώσεις στα ηλεκτρικά σήματα που με την σειρά τους κάνουν αδύνατη την ανάκτηση των δεδομένων στο δέκτη. Ευτυχώς, ο σωστός τερματισμός του διαύλου έχει σημασία μόνο σε εκείνες τις περιπτώσεις όπου τα σήματα που αντιπροσωπεύουν την ψηφιακή πληροφορία μεταδίδονται με ταχύτητες μεγαλύτερες των 57.600 baud και το μήκος του καλωδίου ξεπερνάει τα 500 μέτρα. Όταν ισχύουν τα παραπάνω, οι χρόνοι διάδοσης των ηλεκτρικών σημάτων γίνονται συγκρίσιμοι με την διάρκεια των δυαδικών ψηφίων (το ηλεκτρικό σήμα ταξιδεύει διανύοντας 100 έως 300 μέτρα το κάθε μικροδευτερόλεπτο). Αν το μήκος του καλωδίου είναι μικρότερο από 50 μέτρα και η ταχύτητα μετάδοσης μικρότερη των 57.600 baud τότε, πρακτικά, μπορούμε να χρησιμοποιήσουμε οποιοδήποτε καλώδιο τερματισμένο με αντιστάσεις 120 Ω χωρίς να μας νοιάζει ιδιαίτερα η σύνθετη αντίσταση του.

4.2 Προστασία από υπερτάσεις

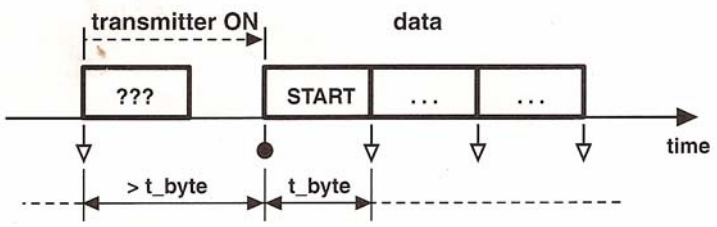
Το κύκλωμα το οποίο συνδέει μια συσκευή RS485 με τον ομώνυμο δίαυλο πρέπει να είναι σχεδιασμένο με τέτοιο τρόπο ώστε να αντέχει σε τάσεις μεγαλύτερες από τις προβλεπόμενες (-7 έως +12). Δυστυχώς όμως τα καλώδια μεγάλου μήκους έχουνε την κακή συνήθεια, να αρπάζουν ηλεκτρομαγνητικό θόρυβο και να τον αναδεικνύουν στα άκρα τους με μορφή παλμών μεγάλου πλάτους. Το γεγονός αυτό

αναγκάζει τους σχεδιαστές να εξοπλίζουν τα κυκλώματα οδήγησης του διαύλους με πρόσθετα εξαρτήματα τα οποία καταπνίγουν τις σύντομες αυτές αιχμές. Εννοείται βέβαια, πως τα εξαρτήματα αυτά δεν παρενοχλούν τη μετάδοση των χρήσιμων δυαδικών σημάτων. Στο κάτω σχήμα φαίνεται ένα τυπικό κύκλωμα βασισμένο σε διόδους zener.



4.3 Μετάδοση δεδομένων

Στο πρώτο σχήμα βλέπουμε, εκτός των άλλων και όλες εκείνες τις αντιστάσεις που είναι απαραίτητες για την ομαλή λειτουργία του διαύλου. Με τη βοήθεια τους οι αγωγοί A και B οδηγούνται σε μια συγκεκριμένη λογική κατάσταση στις περιπτώσεις που όλες οι συνδεδεμένες συσκευές βρίσκονται σε φάση ακρόασης. Τοποθετώντας μια αντίσταση πρόσδεσης είτε προς την γη είτε προς την τάση τροφοδοσίας αναγκάζουμε τα ολοκληρωμένα προσαρμογής να καταναλώνουν περισσότερο ρεύμα, ενώ ταυτόχρονα ευαισθητοποιούμε τον δίαυλο στις πάσης φύσεως ηλεκτρομαγνητικές παρεμβολές (παρατηρείται έντονα όταν οι αγωγοί του διαύλου έχουν μεγάλο μήκος). Μια καλύτερη λύση φαίνεται στο κάτω σχήμα.



Σ' αυτή τη συνδεσμολογία η αρχική στάθμη του διαύλου παραμένει άγνωστη. Μόλις κάποια συσκευή θελήσει να στείλει δεδομένα πρέπει, κατ' αρχήν, να ενεργοποιήσει τα κυκλώματα του ενσωματωμένου πομπού της. Μια τέτοια

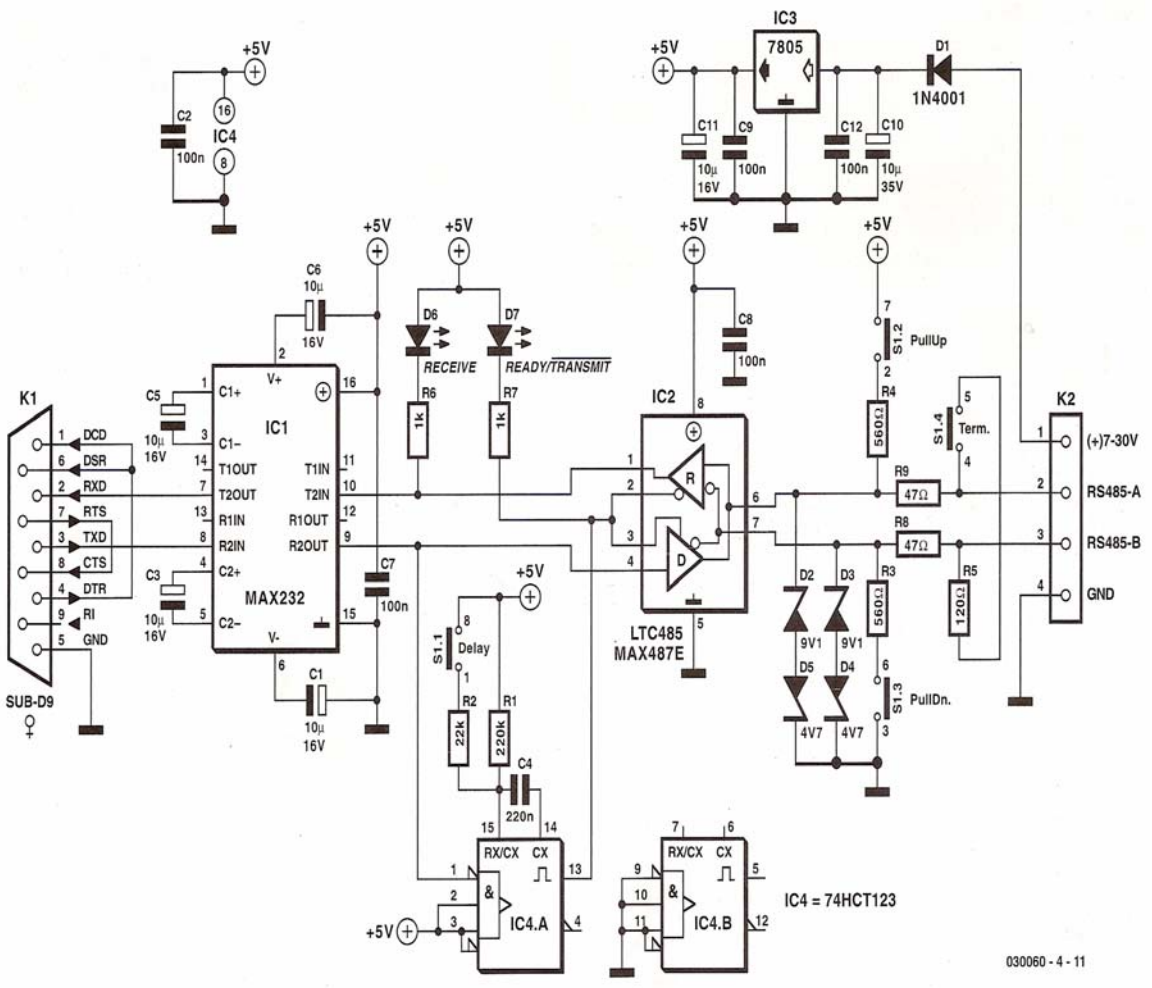
πρωτοβουλία, λόγω ενός ενδεχόμενου κακού τερματισμού, μπορεί να ξεγελάσει τους δέκτες των υπόλοιπων συσκευών κάνοντας τους να πιστέψουν ότι έχουν σταλεί κάποια χρήσιμα δεδομένα. Τα παρασιτικά δεδομένα έχουν αναπαρασταθεί με τα σύμβολα '???'. Για να ελαχιστοποιηθούν οι συνέπειες αυτής της λανθασμένης αντίληψης των δεκτών είναι απαραίτητο αμέσως μετά την ενεργοποίηση να ακολουθεί ένα διάστημα ηρεμίας. Ο νεκρός αυτός χρόνος οφείλει να είναι μεγαλύτερος από το χρόνο που μεταδίδεται ένα κανονικό 'byte' δεδομένων (ισούται με το δεκαπλάσιο του αντίστροφου του ρυθμού μετάδοσης των δυαδικών ψηφίων). Έτσι, αν π.χ. ο ρυθμός μετάδοσης είναι ίσος με 9600 baud, ο χρόνος μετάδοσης αποδεικνύεται ίσος με 1 msec περίπου. Το πλαίσιο δεδομένων ξεκινάει πάντα με ένα ειδικό ψηφίο, το ψηφίο έναρξης (START). Το ψηφίο αυτό ουδέποτε επαναλαμβάνεται κατά το χρονικό διάστημα της μετάδοσης των ψηφίων που συνθέτουν το byte. Επειδή ο δίαυλος RS485 υποστηρίζεται από την φύση του ημιαμφίδρομη επικοινωνία (half – dupex) είναι προφανές πως χρειαζόμαστε ένα πρωτόκολλο με την βοήθειά του οποίου θα ξεκαθαρίζεται εύκολα το πια θα είναι εκείνη η συσκευή που θα επιχειρήσει να στείλει δεδομένα μέσα από αυτόν. Αν δεν καταφύγουμε στις υπηρεσίες ενός έστω και στοιχειώδους πρωτοκόλλου είναι βέβαιο πως θα έχουμε συγκρούσεις δεδομένων που με τη σειρά τους θα οδηγήσουν σε ανεπιθύμητες απώλειες.

4.4 Η σύνδεση του PC στον δίαυλο RS485

Αν τεθούν εντός κυκλώματος οι δύο αντιστάσεις των 560 Ω που σημειώνονται στο πρώτο κύκλωμα, οι πιθανότητες διακίνησης 'παραπλανητικών' byte πρακτικά εκμηδενίζονται. Οι μόνες αιτίες που ίσως καταφέρουν να προκαλέσουν σφάλματα είναι το μήκος του καλωδίου (όταν υπερβαίνει τα 50 μέτρα) ή ο ρυθμός μετάδοσης δεδομένων (όταν υπερβαίνει τα 57600 baud). Ακόμα και τότε όμως, για να σφάλματα θα πρέπει οι αντιστάσεις τερματισμού να απέχουν αισθητά από τον κόμβο σύνδεσης της μονάδας με τους αγωγούς του διαύλου. Δυστυχώς κανένας PC δεν είναι σε θέση να ελέγξει με ακρίβεια των χρονισμό των σειριακών σημάτων που παράγει και κατά συνέπεια να ενεργοποιήσει την βαθμίδα εκπομπής των MAX 487 την ενδεδειγμένη χρονική στιγμή. Ακόμα, οφείλουμε να σημειώσουμε πως, όλοι οι σημερινοί PC χρησιμοποιούν UART με ενσωματωμένη μνήμη FIFO. Η παρουσία της τελευταίας εμποδίζει τον επεξεργαστή του PC να γνωρίζει το πότε ακριβώς εκπέμφθηκε το byte. Το μόνο που ξέρει είναι ότι αυτό το byte έχει τοποθετηθεί στην ουρά εκπομπής. Κατά συνέπεια, το αν ο δίαυλος MAX είναι διαθέσιμος για εκπομπή ή όχι το γνωρίζει μόνο το κύκλωμα του μετατροπέα.

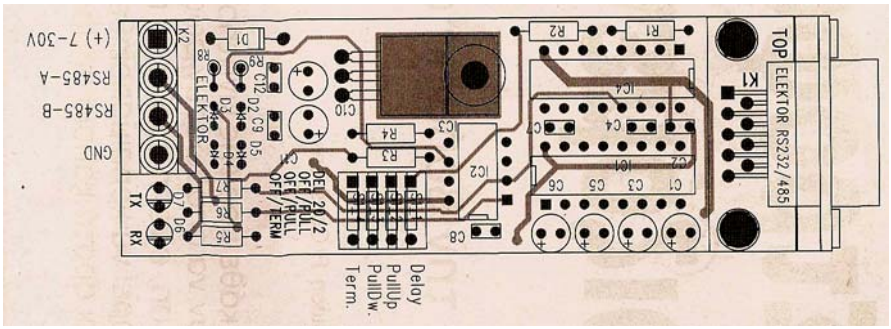
4.5 Ο μετατροπέας

Το θεωρητικό διάγραμμα

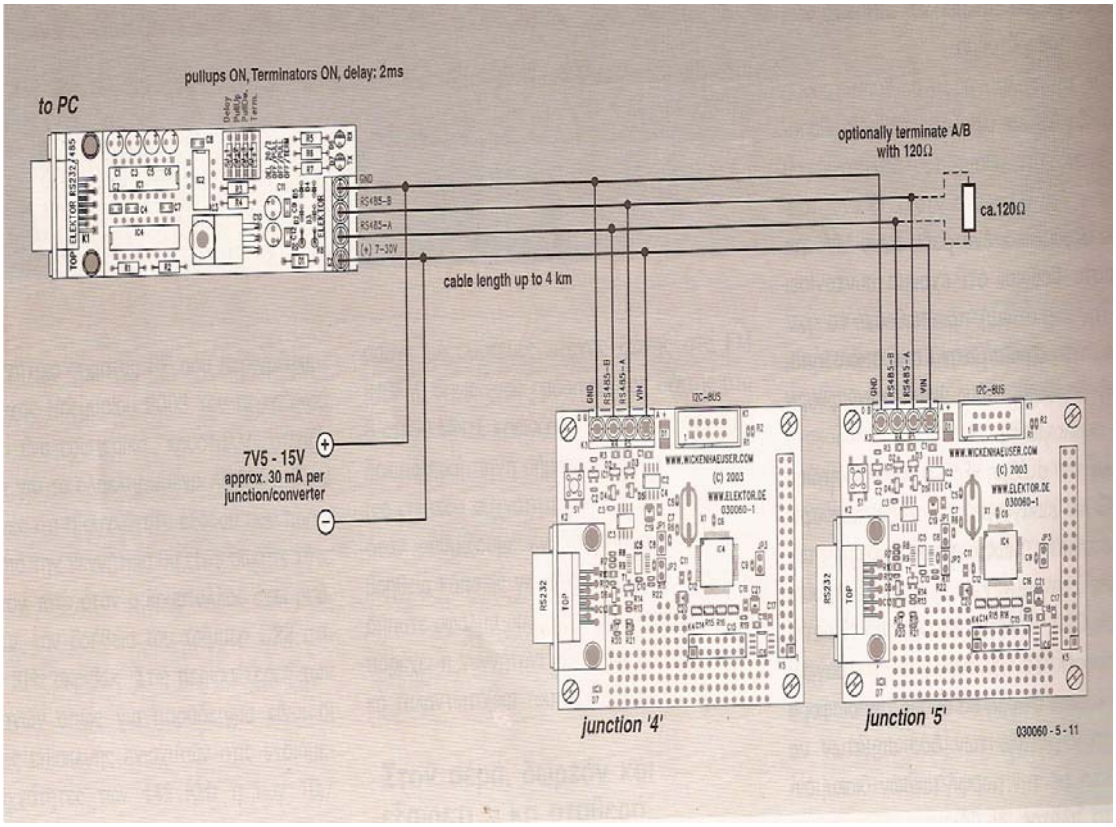


Περιγράφει ένα μάλλον απλό κύκλωμα ικανό να διαχειρίζεται σήματα οποιουδήποτε σειριακού πρωτοκόλλου που μεταδίδονται με οποιουδήποτε ρυθμό μετάδοση. Στην αριστερή μεριά βλέπουμε ένα συνηθισμένο κύκλωμα MAX 232, ενώ στην δεξιά πλευρά το λιγότερο γνωστό MAX 487. Το ρόλο του συνδετικού κρίκου τον παίζουν τα σήματα TTL που αναδεικνύουν και τα δύο ολοκληρωμένα εφαρμόζοντας τα στα εξαρτήματα που βρίσκονται ανάμεσά τους. Τα διαφορικά κωδικοποιημένα σήματα RS485 μεταφράζονται σε TTL και προωθούνται άμεσα στον PC μέσω του MAX 232. Όταν συμβαίνει το αντίθετο, ο επαναδιεγειρόμενος μονοδονητής IC4A θέτει σε λειτουργία τη βαθμίδα εκπομπής του MAX 487 για ένα χρονικό διάστημα ίσο, τουλάχιστον, με το χρόνο αποστολής του πλαισίου των δεδομένων (συνήθως ένα byte). Στη δική μας σχεδίαση μπορεί ένας από τους δύο διαφορετικούς χρόνους 20 msec για ρυθμούς μετάδοσης 1200 baud ή 2 msec για ρυθμούς μετάδοσης 9600 baud. Για τη επιλογή του επιθυμητού χρόνου μπορείτε να αφήσετε ανοικτές ή να ενώσετε αντίστοιχα τις επαφές του ζεύγους 1 του διακόπτη DIP S1 (1-1). ΟΙ υπόλοιποι επιμέρους διακόπτες του S1 επιτρέπουν την σύνδεση της τερματικής αντίστασης των 120 Ω στο σημείο σύνδεσης της πλακέτας με το δίαυλο RS485 (S1-4), ενώ οι άλλοι δύο (S1-2, S1-3) την εξαναγκασμένη οδήγηση των αγωγών του διαύλου σε μια από τις δύο λογικές καταστάσεις όταν όλες οι μονάδες

ηρεμούν (προσθήκη αντιστάσεων 560 Ω). Με τη βοήθεια των παραπάνω διακοπτών, η πλακέτα του μετατροπέα είναι σε θέση να συνδέσει στο ίδιο δίκτυο πολλούς PC και εξίσου πολλές πλακέτες MSC 1210. Τα εξαρτήματα της κατασκευής συναρμολογούνται πάνω στην πιο κάτω πλακέτα



Στο πιο κάτω σχήμα φαίνεται το δίκτυο RS 485 μεταξύ πλακετών και ενός κεντρικού υπολογιστή.



Στην παρούσα συνέχεια θα επικεντρώσουμε το ενδιαφέρων μας στους τρόπους που μπορούν να επικοινωνήσουν οι δικτυωμένες πλακέτες με τον κεντρικό υπολογιστή.

Το λογισμικό δίνει την δυνατότητα χρήσης των πλακετών MSC 1210 σε αρκετές απλές εφαρμογές . Μπορεί όμως εύκολα να θεωρηθεί και σαν το έναυσμα για την ανάπτηξη ευρήτερων εφαρμογών εξίσου χρήσιμων και λειτουργικών. Το πλέον εντυπωσιακό είναι πως αν και βασίζεται στη χρήση των σχετικά πολύπλοκων εντολών print() διαχειριζόμενο ταυτόχρονα αριθμούς κινητης υποδιαστολής καταφέρνει να χωράει σε 5, μόλις K byte μνήμης προγράμματος. Σημειώνουμε πως το μέγαλύτερο μέγεθως κώδικα που είναι σε θέση να παράγει ο μεταγλωτιστής μc/51 , φθάνει τα 8 K byte. Στο πιο πάνω σχήμα βλέπουμε ένα τυπικό διάγραμμα στο οποίο ξεχωρίζουν οι συνδέσεις δύο πλακετών MSC 1210 με την πλακέτα μετατροπής RS 232 / RS 485 που συνδέεται στον κεντρικό υπολογιστή. Το μήκος του καλωδίου πυ συνδέει τον υπολογιστή με πλακέτες-κόμβους πρέπει να είναι μικρότερο των 1000 μέτρων. Όλοι οι κόμβοι τροφοδοτούνται από το κύκλωμα παροχης ρεύματος του μετατροπέα.

4.5.1 Η κατασκευαστική άποψη:

Κάθε μία πλακέτα απορροφά ρεύμα ίσο περίπου με 30 mA. Από τη στιγμή που το ρεύμα αυτό φθάνει στον κόμβο έχοντας διανύσει το καλώδιο σε όλο σχεδόν το μήκος του, πρέπει να λάβουμε υπόψην μας το πρόβλημα της πτώσης τάσης που αναπτύ-σεται πάνω στους αγωγούς του. Για τις ανάγκες της συγκεκριμένης σχεδίασης ορί-
στηκε σαν κανόνας να είναι η τάση τροφοδοσίας κατά 7,5 V μεγαλύτερη από αυτή

που ζητάει η πλακέτα. Θα πρέπει να ληφθεί ακόμα υπόψη η αύξηση των απαιτή-σεων ρεύματος από μια πλακέτα – κόμβο όταν αυτή εκπέμπει μέσα από τα καλώδια του διαύλου. Για τις ανάγκες του συγκεκριμένου κυκλώματος που φιλοξενεί δύο κόμβους, μια αξιόπιστη πηγή τροφοδοσίας θα πρέπει να παρέχει ρεύματα της τάξης των 100 mA. Ας επανέλθουμε όμως στο πρόβλημα της πτώσης τάσης. Αν το μήκος του καλώδίου είναι 1000 μέτρα, τότε το συνολικό μήκος των αγώγων τροφοδοσίας είναι 2000 μέτρα. Θεωρώντας ότι έχουμε περιθώριο να χάσουμε εως και 7,5 V το πολύ, η συνολική αντίσταση των δύο αγωγών θα πρέπει, στη χειρότερη περίπτωση,να είναι 75 Ω (απορροφούμενο ρεύμα είναι 100 mA). Αν χρησιμοποιήσουμε θωρακισμένο καλώδιο τεσσάρων αγωγών διατομής 0,6 χιλιο-τιμή που θεωρείται ικανοποιητικη για την παρούσα εφαρμογή. Μιλώντας όμως γε-νικά, θα λέγαμε πως είναι μάλλον δύσκολη η τροφοδοσία όλων των πλακετών ενος δικτύου μέσω καλωδίου επικοινωνίας. Όπως είχαμε ήδη αναφέρειστο προηγούμενο μερος είναι καλύτερο κάθε πλακέτα – κόμβος να τροφοδοτείται από τη δική της πηγή. Η λύση αυτή είναι καλύτερη από την χρήση καλώδίου με χονδρούς αγώγούς που θα παρουσίαζε μικρότερη πτώση τάσης. Το καλώδιο όμως μεταφέρει και τα χρήσιμα σή-ματα δεδομένων.Για να εξασφαλιστεί η αξιόπιστη μεταδοσής τους πρέπει οιαγωγοί του να τερματίζονται σε ωμικές αντιστάσεις ίσες με την ονομάστικη αντίσταση του κα-
λωδίου. Αν χρησιμοποιηθεί το καλώδιο των τεσσάρων αγωγών που προηγουμένος οι τερματικές αντιστάσεις οφείλουν να έχουν τιμη 120 Ω.

4.6 Το λογισμικό δικτύωσης

Από την στιγμή που εγκατασταθούν τα καλώδια του δικτύου και η αρτιότητα τους, το επόμενο βήμα αφορά σε αυτή καθ' αυτη τη δικτύωση. Το λογισμικό που δινει
Σάρκα και οστά στο συγκεκριμένο δίκτυο RS 485, απαιτεί ένα αξιόπιστο πρωτόκολλο ικανό να διακινεί σειριακούς συρμούς με ταχύτητα 9600 bps (πλαίσια των 8 ψηφίων χωρίς ισοτιμίες και 1 ψηφίο λήξης). Μη ξεχάστείτε και θεωρήσετε οτι η ταχύτητα είναι
57000 bps όπως είναι αυτή μεταξεί του PC και της πλακέτας του μετατροπέα. Κάθε μήνυμα ξεκίνα με τον χαρατήρα '#' και ακολουθείται από την διεύθυνση του παραλήπτη της πληροφορίας. Αμέσως μετά ακολουθούν οι χαρακτήρες που απο-τελούν το περιεχόμενο του μηνύματος σε καθαρή ανάγνωσιμη μορφή (χαρακτήρες ASCII). Το μήνυμα τερματίζεται με μια ακολουθία εντοπισμού πιθαν σφαλμάτων (checksum). Στη δική μας εφαρμογή η έναρξη της ακολουθίας η έναρξη της ακολουθίας εντοπισμού σφαλμάτων σημειώνεται με την βοήθεια του χαρακτήρα '$'.Αμέσως μετά ακολουθούν τα ένα ή δύο byte που μεταφέρουν το άθροισμα των κωδικών ASCII των χαρακτήρων που περιέχονται μεταξύ του χαρακτήρα (#) και του ($). Πάντος για την ανίχνευση των πιθανών σφαλμάτων είναι καλύτερο να χρησιμοποιηθούν κώδικες βασισμένοι σε πολυώνυμα όπως π.χ. CRC-8, CRC-16 κ.λ.π. Αμέσως μετά την ενεργοποίηση της βαθμίδας του πομπού RS 485 πρέπει να μεσολαβήσει ένας 'νεκρός' χρόνος προτού ξεκινήσει η εκπομπή του μηνύματος. Το λογισμικό που τρέχει στο PC εισάγει αυτόματα το χρόνο. Δεν είναι όμως μόνο αυτό που εγγυάται τη σωστή μετάδοση του πρώτου χαρακτήρα. Απαιτούνται και οι αντιστάσεις πρόσδεσης (προς τη γη και την θετική τροφοδοσία), οι οποίες πρέπει οπωσδήποτε να είναι τοποθετημένες. Ο μετατροπέας αρχίζει να δουλεύει μόλις λάβει τον πρώτο χαρακτήρα απο το PC. Ολοι οι συνδετήρες της πλακέτας πρέπει να είναι

τοποθετημένοι εκτός από εκείνον που θέτει εντός κυκλώματος την αντίσταση τερματισμού. Ο βραχυκυκλώτήρας αυτός τοποθετείται μόνο εφ'όσον ο μετατροπέας βρίσκεται στην άκρη του καλωδίου σύνδεσης. Για να μπορέσουμε να ελέγξουμε τη συμπεριφορά του δικτύου κάνοντας χρήση ενός συνηθισμένου προγράμματος εξομοίωσης τερματικού θα αγνοήσουμε την πληροφορία εντοπισμού σφαλμάτων κατα την εκπομπή μηνυμάτων. Η πληροφορία αυτή θα εξακολουθεί να έχει σημασία κατά την φάση της απόκρισης. Το πρόγραμμα επίδειξης που συνοδεύει την κατασκεύη επιτρέπει την εκτέλεση μιας μόνο εντολής, η οποία απεικονίζει στην οθόνη του υπολογιστή την τρέχουσα τιμή της μετρούμενης τάσης. Στην περίπτωση που πάνω στο (μοναδικό) καλώδιο του δικτύου έχουν συνδεθεί περισσότερες από μία πλακέτες, είναι προφανές πως θα πρέπει όλες να τρέχουν το ίδιο πρόγραμμα. Σε μια τέτοια περίπτωση το μόνο που έχουμε να κάνουμε είναι να κατεβάσουμε σ'αυτές το μεταγλωτισμένο πρόγραμμα, έχοντας όμως ορίσει για την κάθε πλακέτα ένα διαφορετίκο αριθμό – ταυτότητα. Στο πηγαίο πρόγραμμα ο αριθμός αυτος είναι ο 5. Από την στιγμή που κάνουμε όλες τις απαραίτητες 'κατεβασιές' μπορούμε να πληκτρολογήσουμε στο PC μας την ακολουθία #5<CR>(με <CR συμβολίζουμε το πλήκτρο ENTER ή RETURN). Το αποτέλεσμα αυτής της κίνησης θα είναι το άναμα του LED της πλακέτας 5 (σ'αυτήν αποεινόταν το μήνυμα) για ένα μικρό χρονικό διάστημα και το αναβόσβημα των LED των υπόλοιπων πλακετών για το ίδιο χρονικό διάστημα. Μετά από αυτή την οπτική επιβεβαίωση της επικοινωνίας θα δούμε να εμφανίζεται στην οθόνη μας η τρέχουσα τιμή του αναλογικού μεγέθους που μετράει η πλακέτα 5. Η ίδια η πλακέτα στέλνει ταυτόχρονα την ίδια τιμή επαυξημένη με περισσότερα στοιχεία στην δική της σειρακη θύρα RS 232, μέσω της οποίας κάτω από συνθηκες 'κατεβαίνει' το πρόγραμμα της εφαρμογής. Μια ανανεωμένη έκδοση του προγράμματος επίδειξης μεταγλωτισμένη με την έκδοση V1.10.12. του μC/51 φιλοξενείται στον κατάλογο...\scr\msr1210\Elmet\Elmet485\ELMFLASH. Σε ότι αφορά την 'κατεβασιά' του ίδιου του μεταγλωτιστή σημειώνουμε πως δεν είναι απαραίτητο να τον κατεβάσουμε μονομιάς (έχει μέγεθος 12 MByte). Μπορούμε να τον κατεβάσουμε τμηματικά από το διδυκτιακό τόπο και να τον συναρμολογήσουμε αργότερα.

4.7 Οδηγίες για την κατάλληλη καλωδίωση ενός RS-485 (TIA/EIA-485-A) δικτύου.

Θα περιγράψουμε την κατάλληλη μέθοδο καλωδίωσης ενός δικτύου RS-485, με συστάσεις για καλώδια twisted-pair (στρεφτού-ζεύγους) και τη σωστή θέση των αντιστάσεων λήξης (termination resistors) . Ως παράδειγμα για τον κατάλληλο και μη κατάλληλο τερματισμό καλωδίων, παρουσιάζονται οι λαμβανόμενες κυματομορφές. Παρουσιάζονται διαμορφώσεις δικτύων για απλά κυκλώματα ενός εκπομπού / πολλαπλών δεκτών (*single-transmitter/multiple receiver*) ,έως πολλαπλού δέκτη (multiple tranceiver) σε κυκλώματα πολλών κλάδων (multi-branched) .

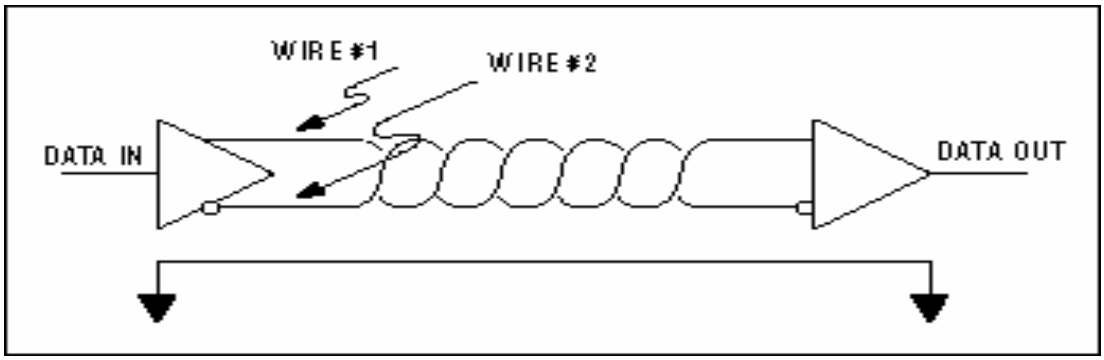
Η προδιαγραφή RS-485 (επίσημα αποκαλούμενη TIA/EIA-485-A) , δεν επεξηγεί ακριβώς πώς ένα δίκτυο RS-485 πρέπει να καλωδιωθεί. Δίνει, όμως, κάποιες οδηγίες. Αυτές οι οδηγίες και οι υγιείς πρακτικές εφαρμοσμένης μηχανικής, είναι η βάση του παρόντος κειμένου. Οι προτάσεις, παρόλα ταύτα, δε συμπεριλαμβάνουν σε καμοιά περίπτωση, όλους τους διαφορετικούς τρόπους σχεδίασης ενός δικτύου.

Το RS-485, διαβιβάζει τις ψηφιακές πληροφορίες μεταξύ πολλαπλών θέσεων. Ο ρυθμός μετάδοσης δεδομένων (datarate) μπορεί, μερικές φορές, να είναι και

μεγαλύτερος από 10Mbps. Το RS-485 σχεδιάστηκε με σκοπό, να διαβιβάζει πληροφορίες σε σημαντικά μήκη (η ικανότητά του φτάνει μέχρι και τα 1000 μέτρα) . Πάντως, η μέθοδος με τις οπτικές ίνες που περιγράφηκε σε προηγούμενη παράγραφο, κάλυπτε μεγαλύτερες αποστάσεις και με μεγαλύτερες ταχύτητες. Η απόσταση και ο ρυθμός δεδομένων, με τα οποία το RS-485 μπορεί να χρησιμοποιηθεί επιτυχώς, εξαρτώνται σημαντικά, από την καλωδίωση του συστήματος.

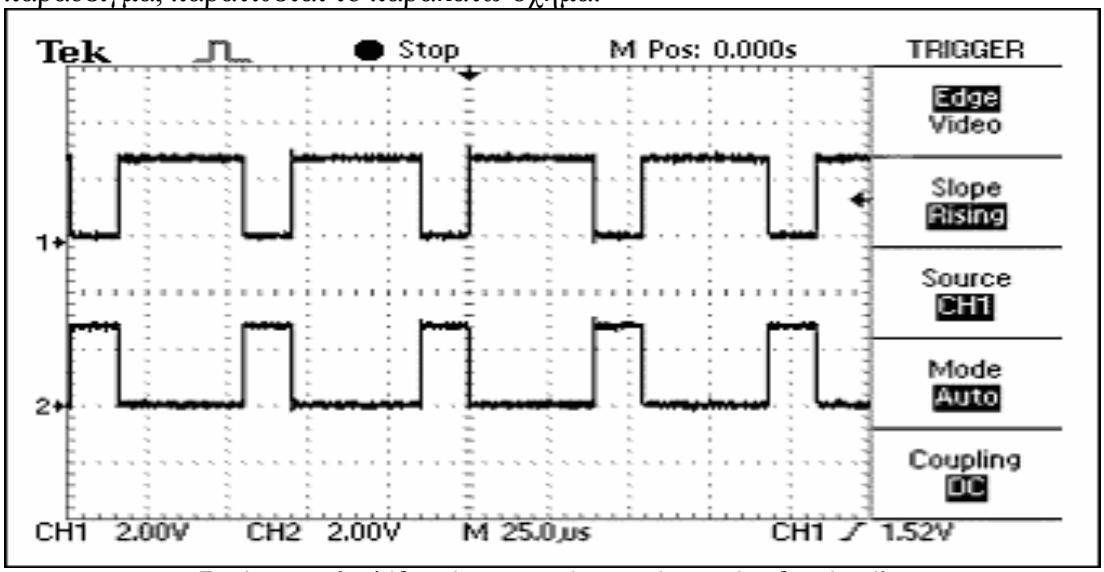
4.8 Καλώδιο

Το RS-485 έχει σαν σκοπό, να είναι ένα ισορροπημένο σύστημα. Θέτοντάς το απλά, υπάρχουν 2 καλώδια εκτός της γείωσης, τα οποία χρησιμοποιούνται για να διαβιβάσουν το σήμα:



Ένα ισορροπημένο σύστημα που χρησιμοποιεί 2 καλώδια εκτός της γείωσης, για να διαβιβάσει τα δεδομένα.

Το σύστημα καλείται ισορροπημένο, επειδή το σήμα στο ένα καλώδιο, είναι το ακριβώς αντίθετο του σήματος, στο δεύτερο καλώδιο. Με άλλα λόγια, εάν ένα καλώδιο διαβιβάζει ‘1’, το άλλο καλώδιο θα διαβιβάζει ‘0’ και αντίστροφα. Ως παράδειγμα, παρατίθεται το παρακάτω σχήμα:



Τα σήματα στα 2 καλώδια ενός ισορροπημένου συστήματος, είναι ιδανικά αντίθετα

Αν και το RS-485 μπορεί να διαβιβάζει επιτυχώς χρησιμοποιώντας πολλαπλούς τύπους μέσων, πρέπει να χρησιμοποιηθεί καλωδίωση του συνήθως αποκαλούμενου "στρεφτού ζεύγους" .

4.9 Τι είναι το στρεφτό ζεύγος, και γιατί χρησιμοποιείται;

Όπως το όνομά του υπονοεί, ένα στρεφτό ζεύγος, είναι απλά ένα ζευγάρι καλωδίων ίσου μήκους και από κοινού στρεφτά. Η χρησιμοποίηση ενός RS-485 για αποστολή σημάτων με καλώδιο twisted-pair, μειώνει δύο σημαντικές πηγές προβλημάτων, για τη σχεδίαση των υψηλής ταχύτητας και μεγάλου μήκους δικτύων: την ακτινοβολία και τη λήψη EMI.

4.9.1 ΕΚΠΟΜΠΗ EMI

Όπως φαίνεται και στο παρακάτω σχήμα, οι μονάδες υψηλής συχνότητας είναι παρόντες οποτεδήποτε οι γρήγορες άκρες, χρησιμοποιούνται στη διαβίβαση των πληροφοριών. Αυτές οι γρήγορες άκρες, είναι απαραίτητες για υψηλοτέρου επιπέδου ταχύτητες, από ότι το RS-485 είναι σε θέση να επιτύχει.

Κυματομορφή ενός τετραγωνικού κύματος 125kHz και του FFT σχέδιό του

Οι υψηλής συχνότητας μονάδες αυτών των γρήγορων άκρων που συνδέονται με τα μακριά καλώδια, μπορούν να δέχονται την επίδραση της ακτινοβολίας EMI. Ένα ισορροπημένο σύστημα που χρησιμοποιεί καλώδιο στριφτού ζεύγους μειώνει αυτήν την επίδραση, με την προσπάθεια να γίνει το σύστημα ένα ανεπαρκές θερμαντικό σώμα. Λειτουργεί με μια πολύ απλή αρχή. Δεδομένου ότι τα σήματα στα καλώδια είναι ίσα αλλά αντίθετα, τα ακτινοβολούντα σήματα από κάθε καλώδιο θα τείνουν επίσης να είναι ίσα και αντίθετα. Αυτό έχει την επίδραση να ακυρώνει το ένα το άλλο, που σημαίνει ότι δεν υπάρχει κανένα ακτινοβολούν EMI. Εντούτοις, αυτό είναι βασισμένο στην υπόθεση ότι τα καλώδια, είναι ακριβώς του ίδιου μήκος και βρίσκονται ακριβώς στη ίδια θέση. Επειδή είναι αδύνατο να υπάρξουν δύο καλώδια στη ίδια θέση συγχρόνως, τα καλώδια πρέπει να είναι όσο το δυνατόν κοντύτερα. Περιστρέφοντας τα καλώδια μεταξύ τους, βοηθάει στην αντίδραση οποιουδήποτε υπολοίμματος EMI, λόγω της πεπερασμένης απόστασης μεταξύ των δύο καλωδίων.

4.9.2 Λαμβανόμενο EMI

Το λαμβανόμενο EMI, είναι βασικά το ίδιο πρόβλημα με το ακτινοβολούν EMI, αλλά σε αντιστροφή. Η καλωδίωση που χρησιμοποιείται σε ένα σύστημα RS-485, θα ενεργήσει επίσης ως κεραία, που λαμβάνει τα ανεπιθύμητα σήματα. Αυτά τα

ανεπιθύμητα σήματα θα μπορούσαν να διαστρεβλώσουν τα επιθυμητά, προκαλώντας στη χειρότερη περίπτωση, σφάλματα δεδομένων. Για τον ίδιο λόγο που το στριφτό ζεύγος αποτρέπει το ακτινοβολούν EMI, θα μειώσει, επίσης, τα ανεπιθύμητα αποτελέσματα του λαμβανομένου EMI. Επειδή τα δύο καλώδια είναι πολύ κοντά το ένα με το άλλο και στρεφτά μεταξύ τους, ο λαμβανόμενος σε ένα καλώδιο θόρυβος, θα τείνει να είναι ο ίδιος με αυτόν που λαμβάνεται στο δεύτερο καλώδιο. Αυτός ο τύπος θορύβου, αναφέρεται ως common mode noise (θόρυβος κοινής λειτουργίας) . Δεδομένου ότι οι RS-485 δέκτες σχεδιάζονται για να ψάχνουν τα σήματα που είναι αντίθετα το ένα προς το άλλο, μπορούν εύκολα να απορρίψουν το θόρυβο, που είναι κοινός και για τα δύο.

5. Χαρακτηριστική σύνθετη αντίσταση του καλωδίου στρεφτού ζεύγους

Ανάλογα με την γεωμετρία του καλωδίου και των υλικών που χρησιμοποιούνται στη μόνωση, το καλώδιο στρεφτού ζεύγους θα έχει μια χαρακτηριστική σύνθετη αντίσταση που σχετίζεται με αυτό, η οποία διευκρινίζεται συνήθως από τον κατασκευαστή του. Η προδιαγραφή RS-485 συστήνει (αλλά δεν υπαγορεύει συγκεκριμένα) , αυτή η χαρακτηριστική σύνθετη αντίσταση να είναι 120 Ohm. Η σύσταση αυτής της σύνθετης αντίστασης, είναι απαραίτητη για τον υπολογισμό του επιπέδου τάσεως υπερφόρτωσης στη χειρότερη περίπτωση και δίνεται στην προδιαγραφή του RS-485. Η προδιαγραφή πιθανώς δεν υπαγορεύει αυτήν την σύνθετη αντίσταση, χάριν ευελιξίας. Εάν για κάποιους λόγους, το καλώδιο 120 Ohm δεν μπορέσει να χρησιμοποιηθεί, συνίστται στη χειρότερη περίπτωση υπερφόρτωσης (ο μέγιστος αριθμός πομπών και δεκτών που μπορούν να χρησιμοποιηθούν) και στη χειρότερη περίπτωση κοινής λειτουργίας, το εύρος τάσης να υπολογίζεται εκ νέου, για να σιγουρευτεί ότι το υπό σχεδίαση σύστημα θα λειτουργήσει.

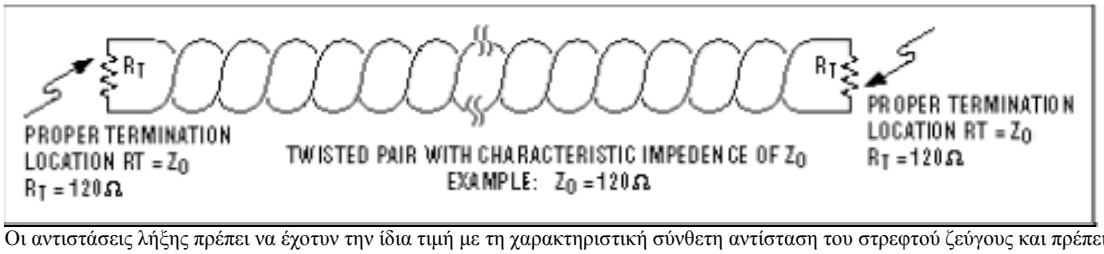
5.1 Αριθμός στρεφτών ζευγών ανά πομπό

Τώρα που έχουμε μια αίσθηση για τον τύπο καλωδίου που απαιτείται, η ερώτηση που προκύπτει είναι ως προς τα πόσα στρεφτά ζεύγη, μπορεί ένας πομπός να οδηγήσει. Η σύντομη απάντηση είναι ακριβώς ένα. Αν και υπό ορισμένες συνθήκες, είναι δυνατό από μια συσκευή αποστολής σημάτων να οδηγηθούν περισσότερα από ένα στρεφτά ζεύγη, αυτό δεν είναι η πρόθεση της προδιαγραφής.

5.2 Αντιστάσεις λήξης

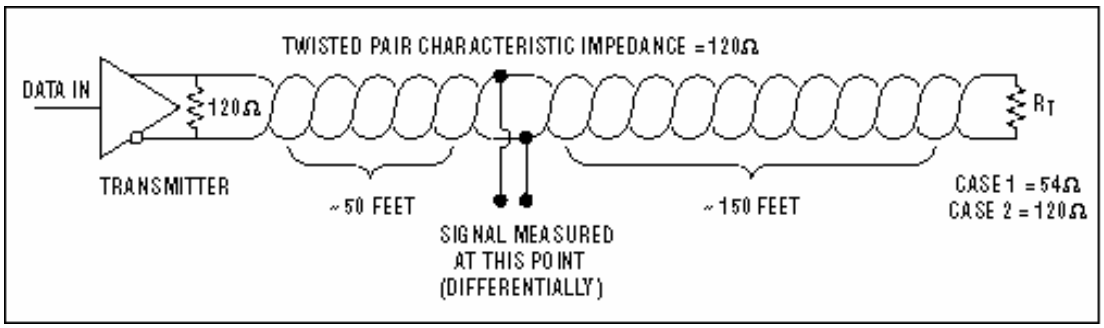
Λόγω των υψηλών συχνοτήτων και των σχετικών αποστάσεων, πρέπει να δοθεί η δέουσα προσοχή, στα αποτελέσματα της γραμμής μετάδοσης. Παρόλα αυτά, μια λεπτομερής συζήτηση των αποτελεσμάτων και οι κατάλληλες τεχνικές λήξης, είναι για τα καλά πέρα από το πεδίο αυτής της περιγραφής. Έχοντας αυτό υπόψην, οι τεχνικές αυτές θα συζητηθούν εν συντομία στην απλούστερη μορφή τους και με τον

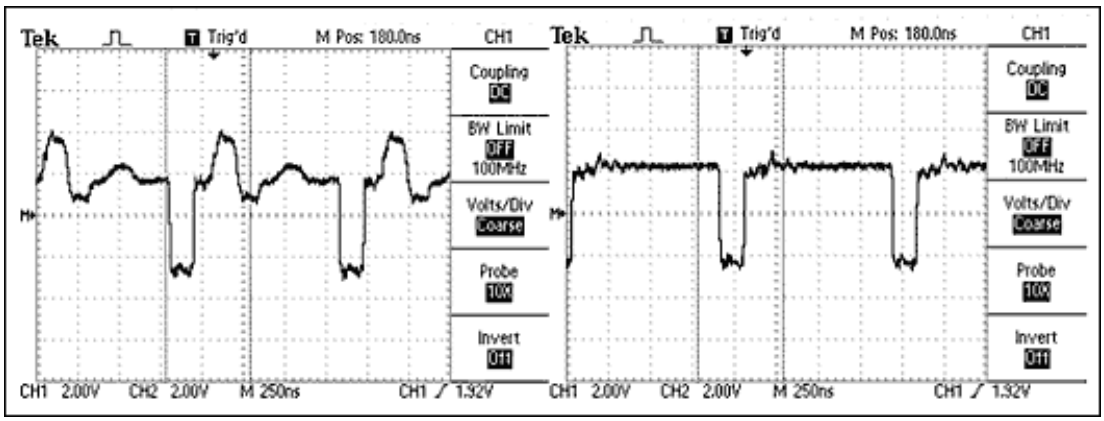
τρόπο που αυτές αφορούν το RS-485. Μια αντίσταση λήξης, είναι απλά μια αντίσταση, που τοποθετείται στην άκρη ή άκρες ενός καλωδίου. Η τιμή αυτής της αντίστασης, είναι υπό ιδανικές συνθήκες η ίδια, με τη χαρακτηριστική σύνθετη αντίσταση του καλωδίου.



Οι αντιστάσεις λήξης πρέπει να έχοτυν την ίδια τιμή με τη χαρακτηριστική σύνθετη αντίσταση του στρεφτού ζεύγους και πρέπει να τοποθετηθούν στις απομακρισμένες άκρες του καλωδίου.

Όταν η αντίσταση λήξης δεν είναι η ίδια με τη χαρακτηριστική σύνθετη αντίσταση της καλωδίωσης, θα εμφανιστούν αντανακλάσεις, δεδομένου ότι το σήμα ταξιδεύει μέσα στο καλώδιο. Αυτό διέπεται από την εξίσωση (Rt-Zo)/(Zo+Rt), όπου Zo είναι η σύνθετη αντίσταση του καλωδίου και Rt, η τιμή της αντίστασης λήξης. Αν και μερικές αντανακλάσεις οφείλονται αναπόφευκτοα στις ανοχές των καλωδίων και των αντιστάσεων, οι αρκετά μεγάλοι και κακοί συνδυασμοί, μπορούν να προκαλέσουν αρκετά μεγάλες αντανακλάσεις και να επιφέρουν σφάλματα στα δεδομένα.





Χρησιμοποιώντας το κύκλωμα που παρουσιάζεται στη κορυφή, η κυματομορφή στα αριστερα λήφθηκε με ένα MAX3485 οδηγώντας ένα 120 Ohm στρεφτό ζεύγος, που τερματίζει με 54 Ohm. Η κυματομορφή στα δεξιά, λήφθηκε με το καλώδιο που τερματίζεται κατάλληλα με 120 Ohm

Έχοντας αυτό υπόψην, είναι σημαντικό η αντίσταση λήξης, να ταιριαστεί με τη χαρακτηριστική σύνθετη αντίσταση, όσο το δυνατόν περισσότερο. Η θέση των αντιστάσεων λήξης είναι επίσης πολύ σημαντική. Οι αντιστάσεις λήξης πρέπει πάντα να τοποθετούνται στις απομακρισμένες άκρες του καλωδίου.

Κατά γενικό κανόνα, οι αντιστάσεις λήξης πρέπει να τοποθετηθούν και στις δύο απομακρισμένες άκρες. Αν και ο κατάλληλος τερματισμός και στις δύο άκρες είναι απολύτως κρίσιμη για τα περισσότερα σχέδια συστημάτων, μπορεί να υποστηριχτεί ότι σε μια μόνο ειδική περίπτωση, απαιτείται μια αντίσταση λήξης. Αυτή η περίπτωση, εμφανίζεται σε ένα σύστημα όπου υπάρχει μια ενιαία συσκευή αποστολής σημάτων, η οποία βρίσκεται στο απομακρισμένο άκρο. Σε αυτήν την περίπτωση, είναι περιττό να τοποθετηθεί αντίσταση λήξης στο άκρο του καλωδίου όπου υπάρχει η συσκευή αποστολής σημάτων, επειδή το σήμα είναι προορισμενο να ταξιδεύει πάντα, μακρυά από αυτό το άκρο.

5.3. Μέγιστος αριθμός πομπών και δεκτών σε ένα δίκτυο

Το απλούστερο δίκτυο RS-485, αποτελείται από έναν ενιαίο πομπό και έναν ενιαίο δέκτη. Αν και χρήσιμο σε διάφορες εφαρμογές, το RS-485 επιτρέπει και μεγαλύτερη ευελιξία, με το να επιτρέψει τους πολλαπλάσιους δέκτες και πομπούς, σε ένα ενιαίο στρεφτό ζεύγος. Ο μέγιστος επιτρεπόμενος αριθμός, εξαρτάται από το πόσο κάθε συσκευή, υπερφορτώνει το σύστημα. Σε έναν “ιδανικό κόσμο” , όλοι οι δέκτες και οι ανενεργοί πομποί, θα έχουν άπειρη σύνθετη αντίσταση και σε καμιά περίπτωση δε θα υπερφορτώσουν το σύστημα. Στον πραγματικό κόσμο, αυτή η περίπτωση δεν είναι εφικτή. Κάθε δέκτης που συνδέεται με το δίκτυο και όλοι οι ανενεργοί πομποί, θα προσθέσουν κάποιο φορτίο. Για να βοηθήσει το σχεδιαστή ενός δικτύου, το RS-485 υπολόγισε ακριβώς πόσες συσκευές μπορούν να προστεθούν σε ένα δίκτυο, δημιουργώντας μια υποθετική μονάδα, αποκαλούμενη "φορτίο μονάδας" (unit load) . Όλες οι συσκευές που συνδέονται με ένα δίκτυο RS-485, πρέπει να χαρακτηριστούν όσον αφορά τα πολλαπλάσια ή τα μέρη των φορτίων μονάδων. Ο μέγιστος αριθμός φορτίων μονάδων που επιτρέπονται σε ένα στρεφτό ζεύγος (υποθέτοντας ένα κατάλληλα τερματισμένα καλώδιο με μια χαρακτηριστική σύνθετη αντίσταση 120 Ohm ή περισσότερο) , είναι 32.

5.4 Οδήγηση οθονών LCD μέσω διασύνδεση I2C

Στην εφαρμογή μας έχουμε την δυνατότητα να εμφανίζουμε κάποια μηνύματα σε μια οθόνη LCD με δύο μόνο σήματα, το SCL και το SDA του διαύλου I2C. Βέβαια τα υλικά έχουν κάποιο κόστος. Το λογισμικό είναι σε μορφή πηγαίου προγράμματος γραμμένο σε μια παραλλαγή της γλώσσας C προσαρμοσμένης στις ιδιαιτερότητες των μικροελεγκτών 8051. Αξιοποιώντας αυτό το πρόγραμμα μέσα από το περιβάλλον της γλώσσας C, θα καταφέρνουμε να απεικονίζουμε οποιαδήποτε μήνυμα στην οθόνη LCD χρησιμοποιώντας την συνάρτηση Icd_printf() . Θα μπορούμε ακόμα να εμφανίζουμε αριθμούς κινητής υποδιαστολής και γενικότερα αριθμούς μεγάλου μεγέθους (Προσομείωση ψηφίων 7 τμημάτων).

Η πλακέτα έχει σχεδιαστεί με τέτοιο τρόπο ώστε να εργάζεται τροφοδοτούμενη από μπαταρίες. Ανεξαρτήτως του τρόπου παροχής ρεύματος, η κατανάλωση της μπορεί να μειωθεί σημαντικά μέσω του λογισμικού το οποίο είναι σε θέση να την σβήνει ανά πάση στιγμή αποσυνδέοντας την, πρακτικά, από την γραμμή των +5 V. Σε αυτή την κατάσταση η κατασκευή απορροφά μόλις μερικά μA. Η διεύθυνση στην οποία ανταποκρίνεται η πλακέτα καθορίζεται με την βοήθεια τριών

βραχυκυκλωτήρων, γεγονός που τις επιτρέπει να αποκτά μέχρι οκτώ διαφορετικές πιθανές τιμές. Αν λάβετε υπ' όψιν ότι το βασικό ολοκληρωμένο της κατασκευής (PCF8574) διατίθεται σε δύο διαφορετικές παραλλαγές που έχουν διαφορετικές βασικές διευθύνσεις, τότε είναι δυνατόν πάνω στο δίαυλο I2C να συνδεθούν έως και δεκαέξι διαφορετικές πλακέτες οθονών. Εναλλακτικά μπορούμε να χρησιμοποιήσουμε την πλακέτα σαν μονάδα παράλληλης εισόδου / εξόδου με την βοήθεια της οποίας θα μπορούμε, για παράδειγμα, να διαχειριζόμαστε πληκτρολόγιο 3 X 4 ή 8 ηλεκτρονόμους.

5.5 Παράλληλη θύρα 8 ψηφίων

Η σύνδεση της πλακέτας μας με οποιαδήποτε άλλη πλακέτα μικροελεγκτή επιτυγχάνεται με την βοήθεια της διπλής σειράς ακίδων που φιλοξενείται κατά μήκος της μεγαλύτερης πλευράς της. Μέσω των ακίδων μεταφέρονται τα απαραίτητα σήματα του διαύλου I2C που με την σειρά τους κάνουν πράξη την επικοινωνία του MSC 1210 με το ολοκληρωμένο μετατροπής PCF8574. Η διεύθυνση του PCF8574. καθορίζεται με την βοήθεια των βραχυκυκλωτήρων που αναφέραμε παραπάνω. Το λογισμικό που έχει γραφτεί για τον MSC 1210 το βλέπει στη δεκαδική θέση 66. Η οθόνη μπορεί να είναι οποιαδήποτε αλφαριθμικού τύπου αρκεί να βασίζεται στον HD44780 της Hitachi (ή κάποιον συμβατό μ' αυτόν). Η διάταξη των ακίδων σε όλες τις οθόνες αυτής της κατηγορίας είναι τις περισσότερες φορές η ίδια. Προσοχή όμως η θέση των ακίδων τροφοδοσίας των LED οπίσθιου φωτισμού είναι συνήθως διαφορετική. Προτού τοποθετήσουμε τη δική μας οθόνη πάνω στην πλακέτα πρέπει να έχουμε ενημερωθεί για την θέση των παραπάνω ακίδων και ενδεχομένως να έχουμε κάνει τις απαραίτητες τροποποιήσεις. Σε γενικές γραμμές, η πλακέτα έχει σχεδιαστεί να δέχεται οθόνες LCD των τεσσάρων γραμμών με είκοσι χαρακτήρες ανά γραμμή. Μπορεί όμως να συνεργαστεί εξίσου καλά και με άλλες μικρότερων δυνατοτήτων. Το μόνο δυσάρεστο που μπορεί να συμβεί με μια μικρότερη οθόνη θα είναι η εμφάνιση μόνο της πρώτης γραμμής του απεικονιζόμενου κειμένου. Και αυτό όμως διορθώνεται εύκολα με μια έξυπνη επέμβαση στο λογισμικό. Η ίδια οθόνη στερεώνεται πάνω στην πλακέτα και σε κάποια απόσταση απ' αυτήν. Η αποκατάσταση των απαραίτητων ηλεκτρικών συνδέσεων επιτυγχάνεται με την βοήθεια μικρών κομματιών καλωδίου. Από λειτουργική άποψη τώρα, σημειώνουμε την παρουσία του τρανζίστορ T2. Με την βοήθεια του η οθόνη τίθεται κυριολεκτικά εκτός λειτουργίας, αφού μέσω αυτού διακόπτεται η τάση τροφοδοσίας της. Εκτός από την εξοικονόμηση ρεύματος που επιτυγχάνεται με αυτό τον τρόπο, έχουμε και ένα ακόμα καλό: ο ελεγκτής της οθόνης εκκινεί πάντα από την αρχή το ενταμιευμένο πρόγραμμα του κάθε φορά που επανατροφοδοτείται. Η επανεκκίνηση του αποδεικνύεται πολλές φορές χρήσιμη, ιδίως όταν επανακαθορίζουμε το περιεχόμενο της εσωτερικής γεννήτριας χαρακτήρων. Η οθόνη επικοινωνιών με το PCF μέσω τεσσάρων γραμμών δεδομένων. Για τον έλεγχο της οθόνης μέσω λογισμικού είναι δυνατόν να χρησιμοποιηθεί ο μεταγλωττιστής της γλώσσας ANCI C μC51.Σε κάθε μία από τις οθόνες LCD που κατά μέγιστο συνδέονται στο δίαυλο I2C μπορούν να οριστούν οι μορφές έως και 8 διαφορετικών χαρακτήρων. Το πρόγραμμα οδηγός τους αξιοποιεί για την απεικόνιση αριθμητικών χαρακτήρων επτά τμημάτων μεγάλου μεγέθους στο κάτω δεξιό μέρος της οθόνης. Επεμβαίνοντας στο ίδιο πρόγραμμα μπορούμε να προσαρμόσουμε ώστε να ανταποκρίνεται στις ανάγκες της δικής μας εφαρμογής.

Το λογισμικό που έχει γραφτεί για την υποστήριξη της κατασκευής είναι πλήρως ιεραρχημένο αποτελούμενο από τρία επίπεδα. Στο υψηλότερο από όλα συναντάμε τη διασύνδεση της εφαρμογής με το χρήστη. Το αμέσως επόμενο αποτελεί μια ενδιάμεση βαθμίδα που μεταφέρει στον οδηγό (χαμηλότερο επίπεδο) του χαρακτήρες που θέλει να εμφανίσει ο χρήστης. Ο οδηγός είναι υπεύθυνος για την επικοινωνία με την οθόνη είτε με δεδομένα εύρους τεσσάρων είτε με δεδομένα εύρους οκτώ δυαδικών ψηφίων.

Οι κυριότερες διαθέσιμες συναρτήσεις είναι οι εξής :

Icd_init()

Αρχικοποιεί την οθόνη και καθορίζει τα τμήματα των χαρακτήρων μεγάλου μεγέθους. Αν η διαδικασία αποδειχθεί επιτυχής επιστρέφει τον χαρακτήρα ‘0’.

Icd_printf()

Κάνει ακριβώς τα ίδια που κάνει και η printf(), μόνο που τώρα ο αποδέκτης των δεδομένων είναι η οθόνη LCD. Η τιμή που επιστρέφει η συνάρτηση συμπίπτει με το πλήθος των χαρακτήρων που ζητήθηκε να απεικονιστούν.

Icd_puttc()

Απεικονίζει ένα μόνο χαρακτήρα.

Icd_Icd_cgchars()

Αντιγράφει ένα μπλόκ 64 bytes από την μνήμη χαρακτήρων της εφαρμογής. Η μνήμη αυτή αποτελεί μέρος της μνήμης προγράμματος του μικροελεγκτή.

Icd_d2_printf()

Ίδια με την printf() μόνο που όλοι οι αριθμοί εμφανίζονται με μεγάλο μέγεθος.

Σημαντικός είναι ο χρονισμός της πλακέτας. Για την καλύτερη δυνατή αξιοποίηση του διαύλου I2C είναι απαραίτητο ο μεταγλωττιστής να γνωρίζει την ταχύτητα του μικροελεγκτή. Η ενημέρωση αυτή επιτυγχάνεται με την βοήθεια της σταθεράς CPU_NSEC, μέσω της οποίας δηλώνεται ο ενεργός χρόνος (σε nsec) που απαιτείται για την εκτέλεση μιας τυπικής εντολής του μικροελεγκτή. Τις περισσότερες φορές ο χρόνος αυτός εκφράζει το μέσο χρόνο εκτέλεσης των εντολών του υπολογιστικού συστήματος. Ο μικροελεγκτής PCF8574 αποτελεί ένα τυπικό κύκλωμα μετατροπής των σειριακών δεδομένων που μεταφέρει ο δίαυλος I2C σε παράλληλη οκταψήφια μορφή και αντιστρόφως. Το PCF8574 διαθέτει οκτώ αμφίδρομες ακίδες στις οποίες αναδεικνύονται byte δεδομένων. Από ηλεκτρική άποψη έχουν τα ίδια χαρακτηριστικά με εκείνα των ακίδων του δημοφιλούς μικροελεγκτή 8051. Όταν εργάζονται σαν έξοδοι χαρακτηρίζονται σαν ανοιχτής ακροής, έχοντας αντιστάσεις πρόσδεσης μεγάλης τιμής, ενώ για να διαβαστούν, όταν εργάζονται σαν είσοδοι, πρέπει πρώτα να έχουν εγγραφεί με λογικό ‘1’. Για την διευθυνσιοδότηση των PCF8574 χρησιμοποιούνται οι ακίδες A0, A1 και A2. Εφαρμόζοντας σε αυτές όλες τις πιθανές λογικές καταστάσεις, καταφέρνουμε να συνδέσουμε στον ίδιο δίαυλο μέχρι και οκτώ ίδια ολοκληρωμένα. Αν πάλι το ζητούμενο είναι η σύνδεση περισσοτέρων, τότε θυμίζουμε πως υπάρχει μια παραλλαγή του ολοκληρωμένου, η PCF8574A, η οποία χαρακτηρίζεται από διαφορετική βασική διεύθυνση. Έτσι το συνολικό πλήθος των παράλληλων θυρών αγγίζει τις δεκαέξι. Το PCF8574 μπορεί να χρησιμοποιηθεί σε πληθώρα εφαρμογών

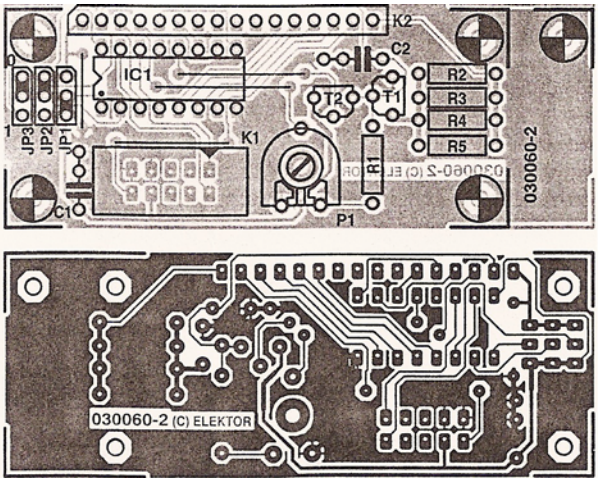
μεταξύ των οποίων ξεχωρίζουν εκείνες που αφορούν στην οδήγηση ηλεκτρονόμων ή τη σάρωση πληκτρολογίων. Για την διευθυνσιοδότηση του PCF8574 αρκεί να σταλεί σε αυτό η παρακάτω λέξη :

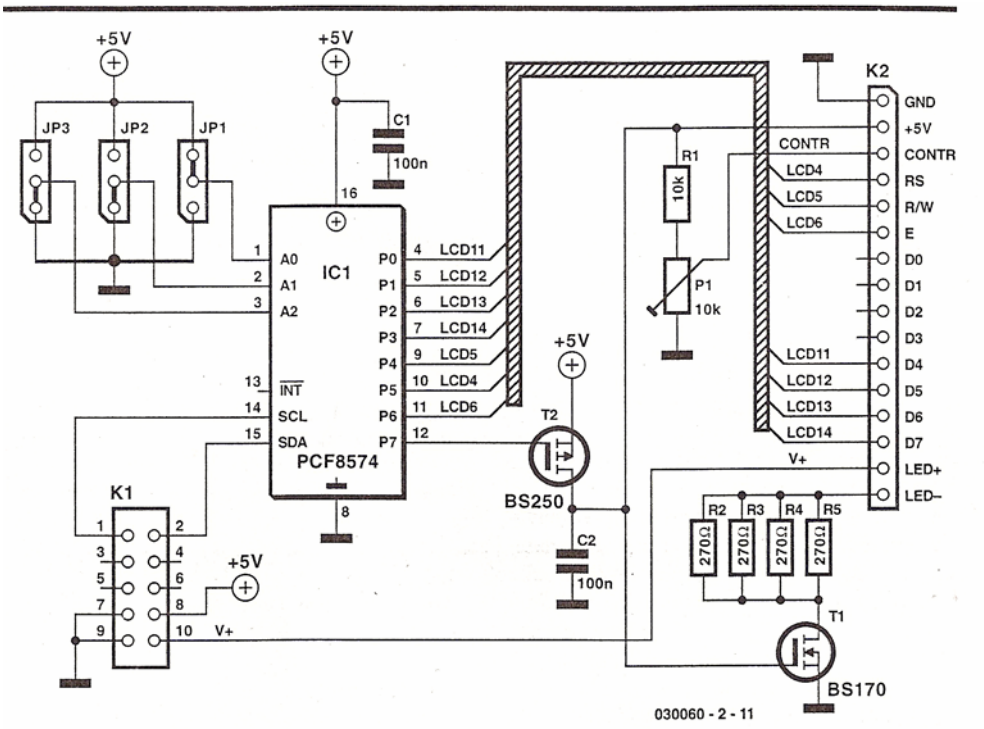
0 1 0 0 A2 A1 A0 0

ενώ για την διευθυνσιοδότηση του PCF8574A :

0 1 1 1 A2 A1 A0 0

Η κατάσταση των ψηφίων αντιστοιχεί με εκείνη των ακίδων του ολοκληρωμένου που θέλουμε να προσπελάσουμε (0 για το δυναμικό της γης, 1 για την τάση τροφοδοσίας).





6. ΚΩΔΙΚΑΣ

****

✓ *M A I N*

```
void main(void){
 uint res;
 uchar i;
 uchar socket;
 xdata uchar *pc;
 int temp100;

 init_msc1210();

 puts("*** www.FLEXGATE.com TCP/IP " __DATE__ ", "__TIME__ " ***");

 // A HTTP-Server with only passive sockets requires only an IP address
 COMPOSE_IP(my_ip,195,251,122,35);

 if(Init_net()) puts("<NET FAILURE>"); // Init Variables, Counter, ...
 printf("My MAC: %02x:%02x:%02x:%02x:%02x:%02x\n",
my_mac[0],my_mac[1],my_mac[2],my_mac[3],my_mac[4],my_mac[5]);
 printf("My IP:  %u.%u.%u.%u\n",my_ip.bytes[0],my_ip.bytes[1],my_ip.bytes[2],my_ip.bytes[3]);

 puts("*** MINI-WEBSERVER - Elektor's ElmFlex ***");
```

```
// Enable ALL (or less) Sockets  as General Server HTTP at port 80
for(i=0;i<MAX_SOCK;i++) SOCKET_SETUP(i,SOCKET_TCP,80,FLAG_PASSIVE_OPEN);

// Initialise A/D at last...
set_adval_bip(0x78,EVREF|EBUF|VREFH|GAIN_8); // Vref on 2.5V, Buff on, BOD off, PGA 1

// * Now Net is ready to start *
for(;;){

 // ENOUGH TIME IN THIS LOOP FOR OTHER JOBS...
 WDSERV(); // Serve Watchdog about every 3 seconds! (see elmet.h)

 res=poll_webserver();

 if((res&0xFF00) ==EVENT_HTTP_REQUEST){
  socket=(uchar)res;

  pc=webpage_name(); // get name of requested page

  printf("Request '%s'\n",pc); // Show requested page (enable for Debug)

  switch(*pc){
  case 'r':
   id_num++; // Increment ID
   sprintf(id,"%u",id_num);
   for(;;){ // Reply-Page: Parse Arguments
    i=url_getarg_no();
    if(!i) break;
    pc=url_getarg_str();

    printf("Arg A%u: '%s'\n",i,pc); // Opt. Show Args... (Enable for Debug)

    switch(i){
    case 1:
     rtc_time.hr=(uchar)atoi(pc);
     break;
    case 2:
     rtc_time.min=(uchar)atoi(pc);
     break;
    case 3:
     rtc_time.sec=(uchar)atoi(pc);
     set_rtc_time();
     break;
    case 4: // Hidden Argument comes first...
     LED3=1; // OFF
     LED4=1; // OFF
     ls3[0]=0; // Empty String
     ls4[0]=0; // Empty String
     break;
    case 5:
     ls3[0]='d'; // 'd' gives checked
     LED3=0; // ON
     break;
    case 6:
     ls4[0]='d'; // 'd' gives checked
     LED4=0; // ON
     break;

    }
    // ignore other Args...
   }

   webpage_bind(socket,reply);
   break;

  case 's':
   sprintf(id,"%u",id_num);
   get_rtc_time();
   sprintf(hr,"%02u",rtc_time.hr);
   sprintf(min,"%02u",rtc_time.min);
   sprintf(sec,"%02u",rtc_time.sec);
   webpage_bind(socket,set);
   break;

  case 't':
```

```
   sprintf(id,"%u",id_num);
   get_rtc_time();
   sprintf(ctime,"%02u:%02u:%02u",rtc_time.hr,rtc_time.min,rtc_time.sec);
   temp100=get_temp();
   sprintf(t_deg,"%d.%02u",temp100/100,((uint)temp100)%100);
   if(temp100<1000) temp100=1000;
   else if(temp100>3499) temp100=3499;
   sprintf(t_wid,"%u",(temp100-1000)/5); // 0..500: Full-Scale
   webpage_bind(socket,t_disp);
   break;

  default:
   sprintf(hits,"%u",++hits_num);
   sprintf(id,"%u",id_num);
   webpage_bind(socket,home);

  }



 }else if(res==EVENT_SOCKET_IDLETIMER){
          RED_LED^=1; // *** BLINK LED ***
       }

 } // for(;;)
}

// EOF
```

✓ *Elmet.c*

```
#include <stdio.h>
#include <irq52.h>


#include <REG1210.H> // SFR
#include <ROM1210.H> // ROM Routines
#asm
.include <REG1210.DEF> // SFR for ASM
#endasm

#include "utility.h" // Utility functions for accessing the A/D
#include "elmet.h" // special ELMET routines and constants

static near long sample;  // ad-value-bypass, valid if semaphore is lit
static near UTIME cnt_time={0,0,0}; // System Timer
near UTIME rtc_time;   // Global Variable

//#define AD_FILT8 // if defined, Summation is used instead of direct samples!

#define ONEMS 11059  // XTAL-1 ms
#define DECIMATION (uint)1440 // 720: 20 Hz data rate (max. 2047)
#define RCAP2 65500  // T2: 9600 Bd

//***** Set sepecial configuration registers HCR0/1 *****
#asm
 .segment _hcr,org $807E
 .dc.b $FC, $FF ; HCR1,HCR0 ; Allow external Memory Access (required for LAN)
#endasm


/*************************************************************************
* set_adval_bip(): Set MUX, Buffers etc.. and init a self calibration.
* Bipolar mode must be set! Note: After self-calibration and/or MUX-channel
* change the first * n Samples must be discarded.
* Retrieve the values with
* if AD_FILT8 is defined, the summation register will be used to
* average 8 samples
*************************************************************************/
void set_adval_bip(unsigned char chan, unsigned char amode){
```

```
#ifdef AD_FILT8
 SSCON=0;
 SSCON=0xdb;
#endif
 ADCON0=amode;
 ADMUX=chan;  // Set new MUX
 ADCON1 = 0x31;  // bipolar, self calibration, offset, gain, sinc3
}

/***************************************************************************
* get_adval_bip(): Get one sample
***************************************************************************/
long get_adval_bip(void){
 unsigned long res;
 EAI=0;    // Disable Auxiliary IRQ
 res=sample;
 EAI=1;    // Re-Enable Auxiliary IRQ
 return res;
}

/***************************************************************************
* get_rtc_time(): Copy static (IRQ maintained) time to user-variable
* The structure is copied by components (although uC/51 allows structure
* copy (like rtc_time=cnt_time) but for small structures this is faster:
***************************************************************************/
void get_rtc_time(void){
 EAI=0;    // Disable Auxiliary IRQ
 rtc_time.hr=cnt_time.hr; // Copy components
 rtc_time.min=cnt_time.min;
 rtc_time.sec=cnt_time.sec;
 EAI=1;    // Re-Enable Auxiliary IRQ
}

/***************************************************************************
* set_rtc_time(): Copy user-time to static (IRQ maintained) time.
* user_s time is verified, on success return 0, else time will not be set.
***************************************************************************/
uchar set_rtc_time(void){
 if(rtc_time.hr>23 || rtc_time.min>59 || rtc_time.sec>59) return 1; // FAIL

 EAI=0;    // Disable Auxiliary IRQ
 cnt_time.hr=rtc_time.hr; // Copy components
 cnt_time.min=rtc_time.min;
 cnt_time.sec=rtc_time.sec;
 EAI=1;    // Re-Enable Auxiliary IRQ
 return 0; // OK
}



// We don't want to have debug info in the interrupt, additionaly only bank 0 is allowed
#pragma option -g0 cpu51 -b0

// Install a vector to the function for Timer 0
IRQ_VECTOR(aux_irq,0x33) // uC/51 special: Bind IRQ to function

/***************************************************************************
* IRQ: Auxiliary interrupt: A/D-Conversion and Timer
*
* If the MACRO AD_FILT8 is defined, the MSC's  8-Sample-Average-Hardware
* Filter will be used. Thus giving an even better analog resolution...
***************************************************************************/
void aux_irq(void) interrupt{
#ifdef AD_FILT8
 if(AIE & 64){  // Check ESUM
  sample=summer(); // get averaged analog value
  SSCON=0;  // Setup new average
  SSCON=0xdb;
 }
#else
 if(AIE & 32){  // Check EADC-Bit
  sample=bipolar(); // get analog value directly
 }
#endif
 if(AIE & 128){  // Check ESEC-Bit
```

```
#asm
  mov A,SECINT // A dummy read of the SECINT clears the Interrupt
#endasm
  if(++cnt_time.sec==60){ // Force Dummy read of SECINT...
   cnt_time.sec=0;
   if(++cnt_time.min==60){
    cnt_time.min=0;
    if(++cnt_time.hr==24){
     cnt_time.hr=0;
    }
   }
  }
 }

 AI=0; // Clear Aux INT (Bit in EICON)
}

#pragma option -g cpu51 -b0- // Restore debug info level to default and enable temporaries

/*********************************************************************
* Because the Adr. Space 0..$7FFF is decoded for the CS8900A,
* the internal XRAM of the CPU must be mapped to >$8000 prior to
* the C startup()-function (startup() expects the RAM to be found at the
* 'correct' location, for initialisation)
*********************************************************************/
#asm
 .segment ___startup_first
 .export __startup_first
 .export STARTUP_FIRST  ; This two lines generate a global linker definition
 STARTUP_FIRST=1  ;  if anything <>0, _startup() will the users call __startup_first()
__startup_first:   ; as first action (see 'lib\lib_c\startup.c').
 mov MCON,#1  ; Map RAM to $8400
 ret
#endasm


/*********************************************************************
* init_msc1210(): Initialize CPU 11.0592 MHz
*********************************************************************/
void init_msc1210(void){
 T2CON  = 0x34;   // T2 as baudrate generator
 RCAP2H  = (RCAP2>>8);
 RCAP2L  = (RCAP2&255);
 SCON   = 0x50;   // Async mode 1, 8-bit UART, enable rcvr, TI=0, RI=0

 MSECH=(ONEMS>>8); // Milisecond register
 MSECL=(ONEMS&255);
 HMSEC=99; // 100 Milisecondregister

#ifdef USE_WD // Defined from extern (see elm_flex.h)
 PDCON &= ~0x04; // WD ON
 WDTCON=0x80; // WD-Enable
 WDTCON=0x00;
 WDSERV(); // initial serve Watchdog
#endif

 // A/D-Converter
 PDCON &= 0x0f7;  //ADC-ON
 ACLK = 11;    // ACLK = 11.0592MHz/(11+1)= 0.9216MHz
 ADCON3=(DECIMATION>>8);
 ADCON2=(DECIMATION&255);  // AD-Speed
 ADCON0 = EVREF|EBUF|GAIN_1;   // Vref on 1.25V, Buff on, BOD off, PGA 1

 // Sec Timer
 PDCON &= 0x0fD;  // System Timer ON
 SECINT=9|128; // Load immediatelly 1/sec

 // IRQ enable (Dont't OR the AIE, else unwanted IRQs may be enabled!)
#ifdef AD_FILT8
 AIE=0x80+0x40;   // Enable SEC_INT(0x80), SUMMER_INT (0x40);
#else
 AIE=0x80+0x20;   // Enable A/D_INT(0x20), SEC_INT(0x80);
#endif
 EAI=1;    // Enable Auxiliary IRQ
```

```
 // Fast XRAM
 CKCON=0;   // NO Stretch of XRAM Access...

}


// EOF
```

✓ *Utility.S51*

```
.include <reg1210.def>

.export _unipolar, _bipolar, _summer

 .segment __unipolar
;;;;;;;;;;;;;;;;;;;;
; unsigned long unipolar(void)
; return the 3 byte adres to R4567 (MSB~LSB)
; unsigned long int with R4=0 (from ADC-Register, if ready)
_unipolar:
 mov a,AIE
 jnb ACC.5,_unipolar
 mov r4,#0
 mov r5,ADRESH
 mov r6,ADRESM
 mov r7,ADRESL
 ret

 .segment __bipolar
;;;;;;;;;;;;;;;;;;;;
; signed long bipolar(void)
; return the 3 byte adres to R4567 (MSB~LSB)
; return signed long int with sign extendsion on R4 (from ADC-Register, if ready)
_bipolar:
 mov a,AIE
 jnb ACC.5,_bipolar
 mov r4,#0
 mov a,ADRESH
 mov r5,a
 mov r6,ADRESM
 mov r7,ADRESL
 jnb ACC.7,positive
 mov r4,#0ffh
positive:
 ret

 .segment __summer
;;;;;;;;;;;;;;;;;;;;
; signed long summer(void);
; return the 4 byte summation registers to R4567 (MSB~LSB) (from summation register)
_summer:
 mov a,AIE
 jnb ACC.6,_summer
 mov r4,SUMR3
 mov r5,SUMR2
 mov r6,SUMR1
 mov r7,SUMR0
 ret
 .end
```

✓ *web_serv.c*

```
#include <stdio.h>
#include <string.h>

#include <reg535.h>

#include "net.h"  // Basic network handling (public)
```

```
#include "web_serv.h"  // Webserver

// This is the Default Message. Using code 200 displays OUR page, whereas 404 may display a Browser's message...
code uchar html_notfound[] = {
"HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\n"
"Connection: close\r\n"
"\r\n"
"<html><head><title>FlexGate</title></head>"
"<body text=\"#0000FF\" bgcolor=\"#FFFF80\" link=\"#FF0000\">"
"FlexGate - Page not found"
"</body></html>\r\n"
};

// Here the HTTP state machine is managed
HTTP_INFO http_info[MAX_SOCK];

xdata uchar *web_args; // static internal variable, points to start of first argument

uchar csock=0xFF; //   static internal variable, Current-Socket Index if !0xFF

/*****************************************************************************
* uint gendyn_data(xdata HTTP_INFO *pinfo, xdata uchar *pbuf)
*
* Fill given buffer with dynamic data, return size (max MAX_TX)
*****************************************************************************/
uint gendyn_html(xdata HTTP_INFO *pinfo, xdata uchar *pbuf){
 uint leftos;  // Left to send or process
 uint cnt;  // No. of data to send
 uint rlen;

 xdata uchar *dsrc;

 code uchar *psrc; // Source is in pinfo
 uchar df;
 uchar c;

 psrc=pinfo->pweb;
 leftos=pinfo->weblen;
 df=pinfo->dyna_flag;
 cnt=0;
 while(leftos){
  c=*psrc;
  psrc++;
  leftos--;
  if(df && c==255){ // Dynamic Sentinel

   dsrc= (xdata uchar*)(*(code uint*)psrc);
   rlen=0;
   while(*dsrc++) rlen++; // Calc. Stringlen of variable

   if(rlen>MAX_TX) rlen=MAX_TX; // Clip too long variables to max. segment size

   if(rlen+cnt>MAX_TX){ // Current data + Variable: Too much!
    psrc--;  // Wind back HTML-Template and
    leftos++; // use a new segment...
    break;  // Exit for now
   }

   dsrc= (xdata uchar*)(*(code uint*)psrc);
   psrc+=2; // Eat Address from Template
   leftos-=2;

   // Insert Var. string in HTML
   while(rlen--){
    c=*dsrc++;
    *pbuf=c;
    pbuf++;
    cnt++;
   }


  }else{
   *pbuf=c;
   pbuf++;
```

```
  cnt++;
  }

  if(cnt==MAX_TX) break; // Buffer full
 }

 pinfo->pweb=psrc;
 pinfo->weblen=leftos;

 return cnt;
}


/*******************************************************************************
*  poll_webserver()
*******************************************************************************/
uint poll_webserver(void){
 uint res;
 uint i;
 uchar uci,c;
 uchar sock,state;
 xdata HTTP_INFO *pinfo;

 uint sendlen;  // Temp. Len
 xdata uchar *pbuf; // Temp. Buffer

 res=poll_net();

 // First check if a socket is pending
 if(!res && csock!=0xFF && http_info[csock].html_state==3 &&  !notready_socket_tcp(csock,RDY_4_TX)){
  // Patch EVENT
  res=EVENT_TCP_DATARECEIVED+csock;
  rcv_len=0;
  csock=0xFF;
 }else if(res==EVENT_SOCKET_IDLETIMER){
  for(uci=0;uci<MAX_SOCK;uci++){
   if(http_info[uci].html_state>=3 &&  !notready_socket_tcp(uci,RDY_4_TX)){
    // Patch EVENT (simulate a received 0-size segment)
    res=EVENT_TCP_DATARECEIVED+uci;
    rcv_len=0;
   }
  }
 }

 // Received a TCP_EVENT. Could indicate received data or a closure of the TCP socket
 if(res>=0xF000){
  sock=(uchar)res; // isolate socket index

  // Only process HTTP-Sockets!
  if(uc_socket[sock].local_port!=80) return res;

  pinfo=&http_info[sock]; // Pointer to HTTP_INFO for the current socket

  state=pinfo->html_state;

  // *** RECEIVED DATA ***
  if((res&0xFF00)==EVENT_TCP_DATARECEIVED){
#if 0 // Disabled (enable only for debugging)
   // Show request...
   for(i=0;i<rcv_len;i++){
    c=rcv_buf[i];
    if(c=='\r') putsl("<CR>");
    else if(c=='\n') puts("<LF>");
    else putchar(c);
   }
#endif
   res=0; // This is our new return EVENT (default: nothing)
   // Parse request
   for(i=0;i<rcv_len;i++){
    c=rcv_buf[i];

    switch(state){
    case 0:
     if(c=='G' && i==0){ // Assume a GET
      state=1;
```

```
    // Emit a HTTP-Request
    res=EVENT_HTTP_REQUEST+sock;

    // Set Default Reply to NotFound
    pinfo->pweb=html_notfound;
    pinfo->weblen=sizeof(html_notfound);
    pinfo->dyna_flag=1; // With interpretation! (because HTML)

   }
   break;
  case 1: // Wait for a EOL after chars were found
   if(c=='\n') state=2;
   break;
  case 2: // Rec. new line, CR LF marks end of HTTP_Request
   if(c=='\n') {
    state=3;  // CR LF: End of Request
   }else {
    if(c!='\r') state=1; // ignore all visible chars
   }
  } // switch
 } // for

 // Inform User about page request
 if(res) {
  pinfo->html_state=state; // Keep state
  return res;   // And return (0 or REQUEST) to the user
 }

 // Try to send data for states 3 and 4
 if(state>=3){
  if(!notready_socket_tcp(sock,RDY_4_TX)){
   state=3;
   if(pinfo->weblen){ // There is still something to send
    pbuf=allocate_tx_buf(); // Allocate a buffer
    sendlen=gendyn_html(pinfo,pbuf); // Fill Buffer
    send_socket_tcp(sock, pbuf, sendlen); // Send buffer (safe, beacuse notready()-checked already)
    csock=sock; // Could send something, retry soon!
   }else{ // Manually close only dynamic Pages...
    state=4; // Waiting for close could block other transfers
    if(!notready_socket_tcp(sock,RDY_4_CLOSE)){
     state=0;
     close_socket_tcp(sock);
    }
   }

  }else{ // !notready
   state=4; // Mark socket as pending, try in 500 msec again
  }

 }
 pinfo->html_state=state; // Keep state
 return 0;   // Ignore Webserver maintained events

}else if(res>=0xF800){ // all Events >=0xF800 close TCP Connections.
 pinfo->html_state=0; // ALL other TCP_EVENTS close the socket (ensured by design)
 return 0;  // Ignore this Event, socket is maintained by webserver
}
}
return res;
}

/*******************************************************************************
*  webpage_name(): Return the Name of the requested page.
*
* Attention: Maybe called only ONCE, because terminates URL-String
*******************************************************************************/
xdata uchar * webpage_name(void){
 uchar c;
 xdata uchar *pc=rcv_buf+5; // Kill GET slash (5 chars)
 web_args=0;  // Assume NO Arguments
 rcv_buf[MAX_RX-1]=0; // Prevent Buffer overread
 for(;;){
  c=*pc;
  if(c=='?') {
   web_args=pc+1;
```

```
  break;
 }else  if(c<=' ') break;
 pc++;
}
*pc=0;
return rcv_buf+5;
}

/*****************************************************************************
* uchar url_getarg_no(void)
*
* Will return the index No of an argument or 0 for none. The Value of an argument
* can be retrieved with url_getarg_str();
* All Arguments must be named A1 - A255 or a1-a255
*****************************************************************************/
uchar url_getarg_no(void){
 uchar ano;
 uchar c;
 if(!web_args) return 0;
 c=*web_args++;
 if(c!='a' && c!='A') return 0; // No Arg found!
 ano=0;
 for(;;){
  c=(*web_args++)-'0';
  if(c>9) break;
  ano*=10;
  ano+=c;
 }
 if(c!= ('='-'0')) return 0; // Format error
 return ano;
}

/*****************************************************************************
* xdata uchar url_getarg_str(void)
*
* Will return the value of the last identified argument
*****************************************************************************/
xdata uchar *url_getarg_str(void){
 xdata uchar *ret;
  uchar c;
 ret=web_args;
 for(;;){
  c=*web_args;
  if(c=='&' || c<=' ') break;
  web_args++; // Read over
 }
 *web_args=0; // Terminate string for this argument
 web_args++; // and set to next Arg.
 return ret;
}



/*****************************************************************************
* webpage_bind(uchar socket, code unsigned char *pd)
* This will bind a webpage to a given socket, after a HTTP-Request was received
* for this page.
*****************************************************************************/
void webpage_bind(uchar sock, code unsigned char *pd){
 xdata HTTP_INFO *pinfo;
 pinfo=&http_info[sock]; // Pointer to HTTP_INFO for the current socket

 pinfo->weblen=*((code uint*)pd);
 pd+=2;
 pinfo->dyna_flag=*pd++;
 pinfo->pweb=pd;
}

// EOF
```

- ✓ ***Netutil.c***

```
#include <stdio.h>
#include <reg535.h>

#include "cs8900.h"  // Utilities
#include "netutil.h"  // Utilities


/*****************************************************************************
* uchar net_match_ulong(unsigned long m)
*
* function, that returns 0 only if the nrext read word from the Ethernet matches
* matches a fixed one
*****************************************************************************/
uchar net_match_ulong(unsigned long m){
 if(Read_Frame_long_8900()!=m) return 1;
 return 0; // MATCH!
}
/*****************************************************************************
* uchar net_match_uint(uint m){
*
* function, that returns 0 only if the next read long from the Ethernet matches
* matches a fixed one
*****************************************************************************/
uchar net_match_uint(uint m){
 if(Read_Frame_word_8900()!=m) return 1;
 return 0; // MATCH!
}

/*****************************************************************************
* unsigned int ip_check(xdata uchar* ps, uint len);
* unsigned int ip_check_more(xdata uchar* ps, uint len, uint old_cs);
*
* Calculate an IP Checksum of Xram Block Block ,
* Used Assembler: This is really FAST!
* ip_check_more: Takes a given CS and adds some more bytes. This is necessary
* for TCP-segments (and optional UDP too), if data are not in a continous block...
* ip_check_more is only allowed if previously an even number of bytes was read...
*****************************************************************************/
#asm
 .segment __ip_check
 .export _ip_check, _ip_check_more
_ip_check:  ; Adr: R6:R7, len: R4:R5, tmp: B
 clr A  ; R6:7 working reg. (delayed in R2:3)
 mov R2,A
 mov R3,A

_ip_check_more:  ; Adr: R6:R7, len: R4:R5, tmp: B, old_cs in R2:R3
 mov DPL,R7
 mov DPH,R6
 mov R6,2 ; CS Working register, copy from R2
 mov R7,3 ; R3
 mov B,#0 ;
 mov A,R4 ; omit 0 words len
 orl A,R5
 jz ?csx
 mov A,R5 ; prepare to use 2 djnz
 jz ?cs1
 inc R4
?cs1: movx A,@DPTR
 inc DPTR
 jnb B.0,?csh
 add A,R7
 mov R7,A
 jnc ?cs2
 inc R6
 mov A,R6
 jnz ?cs2
 inc R7
 sjmp ?cs2
?csh: add A,R6
 mov R6,A
 jnc ?cs2
 inc R7
 mov A,R7
 jnz ?cs2
```

```
 inc R6
?cs2: inc B
 djnz R5,?cs1
 djnz R4,?cs1
?csx:
 ret
#endasm

/*****************************************************************************
* void xram_fast_copy(xdata uchar* src,xdata uchar* dest,uint size);
*
* Copy size data in XRAM
*****************************************************************************/
void xram_fast_copy(xdata uchar* src,xdata uchar* dest,uint size);
#asm
 .segment __xram_fast_copy
 .export _xram_fast_copy
_xram_fast_copy:
 ; src in R6/R7
 ; dest in R4/R5
 ; len in R2/R3
 mov A,R3 ; prepare to use 2 djnz
 jz ?xfc1
 inc R2
?xfc1: ; get from source byte
 mov DPL,R7
 mov DPH,R6
 movx A,@DPTR
 inc DPTR
 mov R7,DPL
 mov R6,DPH
 ; write to dest byte
 mov DPL,R5
 mov DPH,R4
 movx @DPTR,A
 inc DPTR
 mov R5,DPL
 mov R4,DPH
 ; loop
 djnz R3,?xfc1
 djnz R2,?xfc1
 ret
#endasm

// END
```

✓ *Net.c*

```
#include <stdio.h>
#include <string.h>

#include <reg535.h>

#include <irq52.h>


#include "cs8900.h"  // CS8900 Register Definitions
#include "net.h"  // Basic network handling (public)
#include "netutil.h"  // Toolbox

/*****************************************************************************
* Private structs
*****************************************************************************/
typedef struct{
 uint vhl_service; // 0x45xx-0x4Fxx
 uint len;
 uint ident;
 uint frags;
 uchar ttl;
 uchar pcol;
 uint checksum;
 IP_ADR sip;
```

```
 IP_ADR dip;
} IP_HDR;

/*****************************************************************************
* Private Definitions (not in net.h)
*****************************************************************************/
#ifdef FLEXGATE
 #define TIMER_FRQ 28  // (exactly 28.125 @ 22.1184 MHz) Timer Frequency in Hz (<512!)
#endif

#ifdef ELMET
 #define TIMER_FRQ 14  // (exactly 14.0625 @ 11.0592 MHz) Timer Frequency in Hz (<512!)
#endif

/************ TCP Soecket states ************************/
//#define TCP_CLOSED  0 // 0 for all: Socket closed (and listen)

// ** Initial Server States
#define TCP_SYNCON  1   // Confirmed an incomming SYN
//#define TCP_EST  2    // Established, Connection OK

// ** Closing
#define TCP_FINSENT  3    // A FIN was sent. Wait for Acknowledge+FIN
#define TCP_FINCON 4    // Confirmed a FIN with FIN+ACK, waiting for last ACK

// ** Client States
#ifdef USE_TCP_CLIENT
 #define TCP_SYNSENT  5   // Arp was Ok, send SYN now
#endif

/************ UDP Socket states ************************/

#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)
/************ ARP_STATES *********************/
 // IMPORTANT: ARP-States numerical > TCP/UDP/OTHER-STATES! because of Final Timeout!
 #define ARPSENT  6   // Client has sent an ARP
 #define ARPREC   7   // Received Reply for this ARP
  //#define UDP_EST  ARPREC  // For UDP Established is the same as ARP Received...

#endif



/* TCP-Option-Flags */
#define TFIN 0x01
#define TSYN 0x02
#define TRST 0x04
#define TPUSH 0x08
#define TACK 0x10
#define TURGE 0x20  // Flag ignored


/*****************************************************************************
* OPTION DEFS: see net.h
*****************************************************************************/

/*****************************************************************************
* MAC-Level data
*
* Set a (default) MAC for THIS node
*****************************************************************************/
MAC my_mac={0xF1, 0xE2, 0xD3, 0xC4, 0xB5, 0xA6 }; // MAC for this machine: M0:M1:M2:M3:M4:M5

MAC remote_mac;  // used as temp.


#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)
 MAC gateway_mac; // optional Gateway for active oen of an "ouside" peer
#endif


/*****************************************************************************
* IP-Header-Level data
*
* Set a IP for THIS node
```

```
*****************************************************************************/
IP_ADR  my_ip;    // IP for this machine (public)
IP_ADR remote_ip;   // Last read IP

#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)
IP_ADR subnet_ip;   // These two IPs require Setup!
IP_ADR gateway_ip;
#endif

IP_HDR hhdr;    // Temporary header for sending IP-data

/*****************************************************************************
* ICMP/ARP-Level data
*
* ICMP is designed for Standard WIN-pings with 0-32 bytes. Enlarge structs if req.
*****************************************************************************/


typedef struct{ // Definition
  IP_ADR sip;
  IP_ADR dip;
  uint pcol; // 6 for TCP, 17 for UDP
  uint len;
} PSEUDO_HDR;

PSEUDO_HDR pseudo_hdr;  // Used for TCP/IP-Checksums



// Same variables for initial examination of incomming frames
typedef struct{ // Size: 20 Bytes
 MAC sender_mac;
 IP_ADR sender_ip;
 MAC target_mac;
 IP_ADR target_ip;
} ARP_INFO; // The informative Part of an ARP message...

typedef struct{ // Size: 40 Bytes
 uchar type;
 uchar icmp_code;
 uint checksum;
 uint ident; // Commonly unused
 uint sequ; // dto.
 uchar data[32]; // large enough for a standard WINDOWS ping...
} PING_INFO;  // A frame for a standard PING

typedef struct{ // Size: 20 Bytes
 uint sport; // Source port
 uint dport; // Destination port

 WORD2_LONG seq; // Sequence ('my pointer');
 WORD2_LONG ack; // Acknowledge ('your pointer')

 uchar hlen; // TCP header len <<2 (==80 without Options)
 uchar flags; // option Flags TFIN-TURGE

 uint window; // window size
 uint checksum; //
 uint urgent; // urgend pointer (commonly unused)
} TCP_HDR;

#ifdef USE_UDP
typedef struct{ // Size: 8 Bytes (Struct. Currently not used)
 uint sport; // Source port
 uint dport; // Destination port

 uint mlen; // MessageLen
 uint checksum; //
} UDP_HDR;
#endif

#define HFRAME_SIZE 40 // Large enough for the biggest header
typedef union{
 ARP_INFO  arp_info; // 2.nd Level
 PING_INFO ping_info; // 2.nd Level
```

```
 TCP_HDR  tcp_hdr; // 3.rd Level, remote IP in remote_ip, rest of IP_HDR known.
#ifdef USE_UDP
 UDP_HDR  udp_hdr; // 3.rd Level
#endif
 uchar bytes[HFRAME_SIZE]; // Bytes of "generic" access
} HFRAME;

// A Frame for temporary usage 2.nd and 3.rd level
HFRAME hframe;

/*****************************************************************************
* The timer, counts down with about 2 Hz
*****************************************************************************/
uchar net_timer; // Temporary value, counts down until by an IRQ
uchar net_service_cnt; // Additional Timer, counts up. twice /sec.

/*****************************************************************************
* The 'official' buffers in XRAM
*****************************************************************************/

// RX-Buffer (1)
uchar rcv_buf[MAX_RX];   // Buffer for receiving data (Mainly HTTP-Header...)
uint rcv_len;    // Size of received data (int)

// TX-Buffers (x)
uchar tx_buffers[TX_BUFFERS][MAX_TX+1];
uchar tx_bufleft=TX_BUFFERS; // Counts left buffers




/*****************************************************************************
* This uC/51 is designed to support a maximum of >8 simultaneous open sockets
*
* ** Only implemented as a fragment until now!
* ** later there will be a bit-mask holding the 'active' sockets
* ** ** Later socket types: SOCKET_NONE(==0), UDP(port), TCP(port), HTTP, TELNET, ...
*
*****************************************************************************/

UC_SOCKET match_socket;  // Temporary matching socket (Work-pad!)

// *** THE SOCKETS ***
UC_SOCKET uc_socket[MAX_SOCK];  // My (User's) Sockets!



#ifdef DEBUG_REC
/*****************************************************************************
* Debugging Stuff: Records sent and receiced frames
*****************************************************************************/

uint rec_no;
typedef struct{
   uchar typ;  // 'R': Received, 'T' Transmitted, 't' Retransmitted, ...
   uint port;
   unsigned long seq;
   unsigned long ack;
   uchar flags;
   uint len;
} REC_FRAME;

REC_FRAME rec_frame[MAX_REC_FRAME];


/*****************************************************************************
* record_frame: Record 1 Frame
*****************************************************************************/
void record_frame(uchar typ, uint port, unsigned long seq, unsigned long ack, uchar flags, uint len){
   xdata REC_FRAME *pr;
   if(rec_no==MAX_REC_FRAME) return;   // FULL!
   pr=rec_frame+rec_no;
   pr->typ=typ;
   pr->port=port;
```

```
   pr->seq=seq;
   pr->ack=ack;
   pr->flags=flags;
   pr->len=len;
   rec_no++;
}
/*****************************************************************************
* Show Frame, return 1  if data available
*****************************************************************************/
uchar show_frame(uint no){
   uchar flags;
   xdata REC_FRAME *pr;
   if(no>=rec_no) return 0;
   pr=rec_frame+no;
   printf("No:%u '%c' P:%u  S:%lu    A:%lu    ",no+1, pr->typ, pr->port,  pr->seq, pr->ack);
   flags=pr->flags;
   if(flags & TFIN) printf("FIN ");
   if(flags & TSYN) printf("SYN ");
   if(flags & TRST) printf("RST ");
   if(flags & TACK) printf("ACK ");

   printf("   L:%u\n",pr->len);
   return 1;   // OK!
}


#endif



/*****************************************************************************
* xdata uchar* allocate_tx_buf(void);
*
* Find a free buffer, if one found, allocate it and return startadress,
* return 0 if none available!
*****************************************************************************/
xdata uchar* allocate_tx_buf(void){
 uchar ui;
 xdata uchar *pbuf=&tx_buffers[0][0];

 if(tx_bufleft) for(ui=0;ui<TX_BUFFERS;ui++,pbuf+=(MAX_TX+1)){
  if(!(*pbuf)) {
   *pbuf=1; // Mark Buffer as allocated
   tx_bufleft--;
   return pbuf+1; // Return Startadress of buffer
  }
 }
 return 0; // Nothing found!
}

/*****************************************************************************
* void free_tx_buf(xdata uchar* pbuf)
*
* Free TX-Buffer if not more required
*****************************************************************************/

void free_tx_buf(xdata uchar* pbuf){
 pbuf--;   // Pointer to Pos 0(Flag)
 if(*pbuf){
  *pbuf=0; // Buffer now free again...
  tx_bufleft++; // One more Buffer free...
 }
}

/*****************************************************************************
* void free_match_socket(void);
*
* Function for state transition to TCP_CLOSED for a socket, ensures freeing of the
* buffers!
*****************************************************************************/
void free_match_socket(void){
 if(match_socket.buf_outsize1){
  match_socket.buf_outsize1=0;
  free_tx_buf(match_socket.p_outbuf1);
```

```
 }
 if(match_socket.buf_outsize2){
  match_socket.buf_outsize2=0;
  free_tx_buf(match_socket.p_outbuf2);
 }
 if(match_socket.buf_outsize3){
  match_socket.buf_outsize3=0;
  free_tx_buf(match_socket.p_outbuf3);
 }
}


#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)

/*****************************************************************************
* void send_request_ARP for a specific Internet
*
* Send an ARP Request for a specific MAC
*****************************************************************************/
void send_request_ARP(unsigned long ipl){
 // puts("<ARP QUERRY>"); // Inform us...

 RequestSend_8900(42);    // Send Reply

 Write_Frame_long_8900(0xFFFFFFFF);  // To Broadcast
 Write_Frame_word_8900(0xFFFF);    // To Broadcast

 Write_Frame_xdata_8900(my_mac,6);   // From US (MAC)

 Write_Frame_word_8900(0x0806);   // ARP!

 Write_Frame_long_8900(0x10800);   // Ethernet
 Write_Frame_long_8900(0x6040001);  // Request

 Write_Frame_xdata_8900(my_mac,6);   // From US (MAC)
 Write_Frame_long_8900(my_ip.ipl);  // and IP!

 // Variable filled out by Host
 Write_Frame_long_8900(0xFFFFFFFF);  // To Broadcast
 Write_Frame_word_8900(0xFFFF);    // To Broadcast

 // If Our Mask and Destin. Mask differs in the significant netbits, querry MAC of Gateway
 if((ipl^my_ip.ipl)&subnet_ip.ipl) {
  Write_Frame_long_8900(gateway_ip.ipl); // and IP! (far connection over gatewy)
 }else{
  Write_Frame_long_8900(ipl);   // and IP! (local connection)
 }
}
#endif

/*****************************************************************************
* void process_ARP(void){
*
* 2.nd-Level-Multiplexer
* process an ARP request or (not implemented until now) an ARP reply
*****************************************************************************/
uint process_ARP(void){
#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)
    xdata UC_SOCKET *psock;
    uchar ui;
#endif

 uint type;
 if(net_match_ulong(0x10800)) return EVENT_ARP_UNKNOWN; // No ARP!
 if(net_match_uint(0x604)) return EVENT_ARP_UNKNOWN; // No ARP!
 type=Read_Frame_word_8900();
 Read_Frame_xdata_8900(hframe.bytes,20); // Read informative part of ARP message
 if(type>2) return EVENT_ARP_NOTYPE; // Unknown Reply

 if(type==1){ // ARP Request!
  if(hframe.arp_info.target_ip.ipl!=my_ip.ipl) return EVENT_ARP_OTHER; // ARP, but not for us...
  // puts("<ARP request>"); // Inform us...
  RequestSend_8900(42);   // Send Reply

  Write_Frame_xdata_8900(remote_mac,6);  // Kick packet back...
```

```
  Write_Frame_xdata_8900(my_mac,6);  // From US (MAC)
  Write_Frame_word_8900(0x0806);  // ARP!

  Write_Frame_long_8900(0x10800);
  Write_Frame_long_8900(0x6040002); // Response

  Write_Frame_xdata_8900(my_mac,6);  // From US (MAC)
  Write_Frame_long_8900(my_ip.ipl); // and IP!
  Write_Frame_xdata_8900(hframe.bytes,10);  // Kick half packet back...
  return EVENT_ARP_REQUEST;    // No Event of interest, but an EVENT

 }else{ // Arp response! For us?
#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)
  //  puts("<ARP response>");
  // Will work with all non-0 type sockets!
      psock=uc_socket;
      for(ui=0;ui<MAX_SOCK;ui++,psock++){
          if(psock->socket_type && psock->state==ARPSENT){
    // Only ARP-Sockets are of interest if an offered ip is matched
    if((psock->sremote_ip==hframe.arp_info.sender_ip.ipl) || // Either direct IP match
      // Or Response is from Gateway, if subnets differ
       ((  (psock->sremote_ip ^ my_ip.ipl) & subnet_ip.ipl  )&&(hframe.arp_info.sender_ip.ipl==gateway_ip.ipl))  ){

     // Copy MAC
     xram_fast_copy(hframe.arp_info.sender_mac,psock->sremote_mac,6);
     psock->state=ARPREC;

     psock->retry_cnt=0;
     psock->timer=1;  // Start NOW!

     return EVENT_ARP_OURREPLY;

    }
   }
  }
#endif
  return EVENT_ARP_OTHERREPLY;
 }
}

/*****************************************************************************
* void process_ICMP(uint dlen)
*
* 2.nd-Level-Multiplexer
* received an ICMp frame ('PING')
*****************************************************************************/
uint process_ICMP(uint dlen){
 //  MICROCHIP SAYS IT IS SAVE TO TRUNCATE ICMPs, so truncate...
 if(dlen>sizeof(HFRAME)) dlen=sizeof(HFRAME);  // Truncate too long Pings!

 Read_Frame_xdata_8900(hframe.bytes,dlen); // Read Sender's Data
 if(hframe.ping_info.type==0){
  // *** NOT REQUIRED FOR SERVER MODE! ***
  // puts("<ECHO REPLY ???>");
  return EVENT_ICMP_REPLY;

 }else if(hframe.ping_info.type==8){

  //puts("<ICMP ECHO REQUEST>"); // For debugging...

  // Reflect block as reply
  hframe.ping_info.type=0;
  hframe.ping_info.checksum=0;
  hframe.ping_info.checksum=~ip_check(hframe.bytes,dlen);

  // Now, send out reply
  hhdr.vhl_service=0x4500;
  hhdr.len=sizeof(IP_HDR)+dlen;
  hhdr.ident=0;
  hhdr.frags=16384; // No Fragmentation
  hhdr.ttl=100; // Industrial standard
  hhdr.pcol=1; // ICMP
  hhdr.checksum=0;
  hhdr.sip.ipl=my_ip.ipl;
  hhdr.dip.ipl=remote_ip.ipl;
```

```
 hhdr.checksum=~ip_check((xdata uchar*)&hhdr,sizeof(IP_HDR));

 RequestSend_8900(dlen+sizeof(IP_HDR)+14); // Send Reply
 Write_Frame_xdata_8900(remote_mac,6);  // Kick back...
 Write_Frame_xdata_8900(my_mac,6);  // From US (MAC)
 Write_Frame_word_8900(0x800);  // type IP

 Write_Frame_xdata_8900((xdata uchar*)&hhdr,sizeof(IP_HDR));  // Send Header
 Write_Frame_xdata_8900(hframe.bytes,dlen);  // and echo
 // puts("<PING>\a"); // show it!

 return EVENT_ICMP_REQUEST; // Someone has PINGED us!

}else return EVENT_ICMP_UNKNOWN; // Ignore the Rest...
}


#ifdef USE_UDP
/*****************************************************************************
* void process_UDP(void)
*
* 3.rd-Level-Multiplexer
* Process the Header (and contents) of a UDP datagram.
*
* An UPD frame may arrive as broadcast, so treat is as non important first...
*
* Note: It is intended hframe my by used for synthesisinhg a response header...
*
*****************************************************************************/
uint process_UDP(uint dlen){
    xdata UC_SOCKET *psock;
 uint ui;

 uint udp_sport, udp_dport;
 udp_sport=Read_Frame_word_8900();
 udp_dport=Read_Frame_word_8900();
 if(net_match_uint(dlen)) return EVENT_UDP_ERROR; // a simple check for plausibility...
 Read_Frame_word_8900(); // Ignore CS...


    psock=uc_socket;
    for(ui=0;ui<MAX_SOCK;ui++,psock++){
       if(psock->socket_type==SOCKET_UDP){  // Only UDP-Sockets are of interest
           // Test local port match
              if(psock->local_port==udp_dport){
   psock->sremote_port=udp_sport;  // Copy Sender's Sourceport
   psock->sremote_ip=remote_ip.ipl; // Copy Sender's IP
   xram_fast_copy(&remote_mac[0],psock->sremote_mac,6); // Copy Sender's MAC to socket
   dlen-=8; // Subtract header length
   if(dlen>MAX_RX) break;   // Ignore too long frames...
   Read_Frame_xdata_8900(rcv_buf,dlen);  // Read Sender's Data, if any
   rcv_len=dlen;    // remember size of read data...
                    return EVENT_UDP_DATARECEIVED+ui;
                   }
       }
    }

 // ********** Check local sockets for a match or return ..... ************
 return EVENT_UDP_UNSOLICITED; // None of our Sockets: RETURN
}

/*****************************************************************************
* void send_upd();
*****************************************************************************/
void send_upd(xdata char* data, uint len,xdata MAC *pmac,unsigned long rem_ipl,uint sport, uint dport){
 // Now fill out IP-Header
 hhdr.vhl_service=0x4500;
 hhdr.len=sizeof(IP_HDR)+8+len; // 8 Bytes UDP-Header
 hhdr.ident=0;
 hhdr.frags=16384; // No Fragmentation
 hhdr.ttl=100; // Industrial standard
 hhdr.pcol=17; // UDP
 hhdr.checksum=0;
 hhdr.sip.ipl=my_ip.ipl;
 hhdr.dip.ipl=rem_ipl;
```

```
 hhdr.checksum=~(ip_check((xdata uchar*)&hhdr,sizeof(IP_HDR))); // IP-Header only

 // Now, send out reply
 RequestSend_8900(sizeof(IP_HDR)+14+8+len); // Send Reply:  ETHERNET_HDR IP_HDR UDP_HDR +(data)
 Write_Frame_xdata_8900((xdata uchar*)pmac,6);  // Physical destination
 Write_Frame_xdata_8900(my_mac,6);  // From US (MAC)
 Write_Frame_word_8900(0x800);  // type IP

 Write_Frame_xdata_8900((xdata uchar*)&hhdr,sizeof(IP_HDR));  // Send IP Header
 Write_Frame_word_8900(sport);
 Write_Frame_word_8900(dport);
 Write_Frame_word_8900(len+8);  // Including UDP_HDR...
 Write_Frame_word_8900(0);  // 0: Means: Checksum not computed

 Write_Frame_xdata_8900(data,len);   // Send data
}
#endif

/*****************************************************************************
* void send_TCP();
*
* Will send a given Segment as IP-TCP-(DATA). ACK,SEQU,WINDOW,FLAGS must be set
* by the caller! TCP-Checksum is computed. MSS not regarded, because sure to be less
* than the rest of the network...
* Data for header in HFRAME already setup
*****************************************************************************/
void send_TCP(xdata char* data, uint len,xdata MAC *pmac,unsigned long rem_ipl){
 uint data_cs;
 hframe.tcp_hdr.hlen=80;   // Standard Size: 20 Bytes
 hframe.tcp_hdr.checksum=0;

 pseudo_hdr.sip.ipl=my_ip.ipl;  // Built Pseudo-Header for Checksum
 pseudo_hdr.dip.ipl=rem_ipl;
 pseudo_hdr.pcol=6; // TCP
 pseudo_hdr.len=len+20;   // Data+TCP-Header, without Pseudo-header!

 data_cs=ip_check(data,len); // Checksum of Data Block
 hframe.tcp_hdr.checksum = ~( // Checksum of Header, Datablock and Pseudo_header
  ip_check_more(hframe.bytes,20, // TCP-Header
   ip_check_more((xdata uchar*)&pseudo_hdr,sizeof(PSEUDO_HDR), // Pseudo-header
    data_cs))); // Data

 // Now fill out IP-Header
 hhdr.vhl_service=0x4500;
 hhdr.len=sizeof(IP_HDR)+20+len; // 20 Bytes TCP-Header (add MSS if required)
 hhdr.ident=0;
 hhdr.frags=16384; // No Fragmentation
 hhdr.ttl=100; // Industrial standard
 hhdr.pcol=6; // TCP
 hhdr.checksum=0;
 hhdr.sip.ipl=my_ip.ipl;
 hhdr.dip.ipl=rem_ipl;
 hhdr.checksum=~(ip_check((xdata uchar*)&hhdr,sizeof(IP_HDR))); // IP-Header only

 // Now, send out reply
 RequestSend_8900(sizeof(IP_HDR)+14+20+len); // Send Reply:  ETHERNET_HDR IP_HDR TCP_HDR +(data)


 Write_Frame_xdata_8900((xdata uchar*)pmac,6);  // Physical destination

 Write_Frame_xdata_8900(my_mac,6);  // From US (MAC)
 Write_Frame_word_8900(0x800);  // type IP

 Write_Frame_xdata_8900((xdata uchar*)&hhdr,sizeof(IP_HDR));  // Send IP Header
 Write_Frame_xdata_8900(hframe.bytes,20);  // Send TCP Header
 Write_Frame_xdata_8900(data,len);   // Send data

#ifdef DEBUG_REC
   // Record Data of Transmitted Frame
   record_frame('T',hframe.tcp_hdr.dport,hframe.tcp_hdr.seq.u,hframe.tcp_hdr.ack.u, hframe.tcp_hdr.flags,len);
#endif

}
```

```
/*****************************************************************************
* void send_incomming_reset_TCP();
*
* Build reset-segment as reply without using match_socket, i.e. as denial for an
* incomming request... ACK included.
*****************************************************************************/
void send_incomming_reset_TCP(uint dlen,xdata MAC *pmac,unsigned long ipl){
 unsigned long ack;
 uint sport;
 sport=hframe.tcp_hdr.dport;
 hframe.tcp_hdr.dport=hframe.tcp_hdr.sport; // Bounce port
 hframe.tcp_hdr.sport=sport;

 // Window, Flags and Set ACK and SEQU in the response, rest will be completed by send_tcp
 hframe.tcp_hdr.window=0;  // No reply!
 ack=hframe.tcp_hdr.seq.u+dlen;
 if(hframe.tcp_hdr.flags & (TSYN | TFIN)) ack++;
 hframe.tcp_hdr.seq.u=hframe.tcp_hdr.ack.u;
 hframe.tcp_hdr.ack.u=ack;
 hframe.tcp_hdr.flags=TRST+TACK;
 send_TCP(0,0,pmac,ipl); // Replay
}

/*****************************************************************************
* void send_match_ok_TCP();
*
* Build Segment header and send it as regular Header, Data are in *pdata, size alen
* match_socket must fit! hframe used as a temporary variable
*****************************************************************************/
void send_match_ok_TCP(xdata uchar *pdata, uint dlen, uchar flags){
 unsigned long seq;

 hframe.tcp_hdr.sport=match_socket.local_port; // Our Port
 hframe.tcp_hdr.dport=match_socket.sremote_port; // Remote

 // Window, Flags and Set ACK and SEQU in the response, rest will be completed by send_tcp
 hframe.tcp_hdr.window=MAX_RX;  // Sender: Do not send more the MAX_RX

 seq=match_socket.sseq.u;
 if(flags & (TSYN)) seq--;  // If a SYN is sent, count this as 1
 hframe.tcp_hdr.seq.u=seq;

 hframe.tcp_hdr.ack.u=match_socket.sack.u; // This was received from the Sender
 hframe.tcp_hdr.flags=flags;

 // Send empty
 send_TCP(pdata,dlen,match_socket.sremote_mac, match_socket.sremote_ip );

 // printf("<TX P:%u A:%x S:%x F:%u, T:%u> ", hframe.tcp_hdr.dport,
hframe.tcp_hdr.ack.w.l_word,hframe.tcp_hdr.seq.w.l_word,hframe.tcp_hdr.flags,match_socket.state);

}

/*****************************************************************************
* uint state_machine_TCP(uint dlen);
*
* 4.rd-Level-Multiplexer
*
* Process one step in the TCP-state-Machine. The 'match_socket' follows the
* TCP-State-Machine if its type is SOCKET_HTTP or SOCKET_TCP.
* The low_word of the sequence-number is the offset for SOCKET_HTTP. May wrap
* for SOCKET_TCP! So resending the sequence-number $zzzz0000 for SOCKET_HTTP implies
* resending the SYN!
* if this routine is called, destination/source port match already checked and
* 'match_socket' copied...
*****************************************************************************/
uint state_machine_TCP(uint dlen){

 if(hframe.tcp_hdr.flags&TRST){
  free_match_socket();   // Free Buffers if allocated...
  match_socket.state=TCP_CLOSED;  // Connection ends immediatelly
  return EVENT_TCP_RESETRECEIVED;
 }
```

```
 // printf("<rx P:%u A:%x S:%x F:%u, T:%u> ",
hframe.tcp_hdr.sport,hframe.tcp_hdr.ack.w.l_word,hframe.tcp_hdr.seq.w.l_word,hframe.tcp_hdr.flags,match_socket.state);

 match_socket.timer=BASIC_RETRY_TIMER;

 switch(match_socket.state){

 // Socket was listening. Only a SYN could change this
 case TCP_CLOSED:  // Passive open!
  if(!(hframe.tcp_hdr.flags&TSYN)) break;
  // puts("<SYN RECEIVED>");

#ifdef USE_TCP_CLIENT
  if(match_socket.tcp_client_flag!=FLAG_PASSIVE_OPEN) break; // Passove open not allowed.
#endif

  // Fast XDATA copy by two casts... (6 Bytes)
  *(xdata unsigned long*)match_socket.sremote_mac=*(xdata unsigned long*)remote_mac;
  *(xdata uint*)(match_socket.sremote_mac+4)=*(xdata uint*)(remote_mac+4);
  // Save remote's IP, set by process_IP() and other data
  match_socket.sremote_ip=remote_ip.ipl;
  match_socket.sremote_port=hframe.tcp_hdr.sport; // Remote Port match already matching!
  // Our Ack is sender's Sequence!
  match_socket.sack.u=hframe.tcp_hdr.seq.u+dlen+1; // +1: Bec. SYN rcvd.
  match_socket.sseq.w.h_word=net_service_cnt;   // Time ascending...
  match_socket.sseq.w.l_word=0;    // Our relative Pointer (for HTTP)

  send_match_ok_TCP(0,0,TSYN+TACK); // Reply with a single SYN+ACK
  // puts("<SYN+ACK SENT>");

  match_socket.state=TCP_SYNCON;  // SYN confirmed with SYN+ACK
  match_socket.retry_cnt=0;
  return EVENT_TCP_SYNRECEIVED; // Low-Byte added by caller!

#ifdef USE_TCP_CLIENT
 case TCP_SYNSENT:
  // puts("<ACTIVE OPEN SYN-RECEIVED>");
  if(!(hframe.tcp_hdr.flags&TSYN)) break;
  hframe.tcp_hdr.seq.u++;    // Count remote SYN
  match_socket.sack.u=hframe.tcp_hdr.seq.u; // +1: Bec. SYN rcvd.
#endif

 case TCP_SYNCON:
 case TCP_EST:
  if(!(hframe.tcp_hdr.flags&TACK)) break;
  if(dlen>MAX_RX) dlen=MAX_RX;  // IDIOTA! Clip data in size (don't know if this is safe?)

  // Here a small problem is silently ignored: A not acknowled Segment which is restransmitted larger
  // could contain old data as a part (maybe for TELNET...)
  // Silently assume all Segments have valid ACK

  if(match_socket.sack.u!=hframe.tcp_hdr.seq.u) return EVENT_TCP_OUTOFBOUNDS; // Ignore-out-of-bounds segments!

  match_socket.state=TCP_EST;  // Connection now established

  match_socket.sack.u+=dlen;
  Read_Frame_xdata_8900(rcv_buf,dlen);  // Read Sender's Data, if any
  rcv_len=dlen;    // remember size of read data...

  // Matching 3 Sockets? -> Clear ALL
  if(match_socket.buf_outsize3 && hframe.tcp_hdr.ack.u==match_socket.sseq_3){
   //putsl("<M123>");
   free_tx_buf(match_socket.p_outbuf3);
   free_tx_buf(match_socket.p_outbuf2);
   free_tx_buf(match_socket.p_outbuf1);
   match_socket.buf_outsize3=0;
   match_socket.buf_outsize2=0;
   match_socket.buf_outsize1=0;

  // Matching Sockets 2 and 1: Free 1,2, Shift 3 to 1
  }else if(match_socket.buf_outsize2 && hframe.tcp_hdr.ack.u==match_socket.sseq_2){
   //putsl("<M12>");
   free_tx_buf(match_socket.p_outbuf2);
   free_tx_buf(match_socket.p_outbuf1);
```

```
 match_socket.sseq_1=match_socket.sseq_3;
 match_socket.p_outbuf1=match_socket.p_outbuf3;
 match_socket.buf_outsize1=match_socket.buf_outsize3;

 match_socket.buf_outsize2=0;
 match_socket.buf_outsize3=0;

// Matching Sockets 1 Free 1, Shift 2 to 1, 3 to 2
}else if(match_socket.buf_outsize1 && hframe.tcp_hdr.ack.u==match_socket.sseq_1){
 //putsl("<M1>");
 free_tx_buf(match_socket.p_outbuf1);

 match_socket.sseq_1=match_socket.sseq_2;
 match_socket.p_outbuf1=match_socket.p_outbuf2;
 match_socket.buf_outsize1=match_socket.buf_outsize2;

 match_socket.sseq_2=match_socket.sseq_3;
 match_socket.p_outbuf2=match_socket.p_outbuf3;
 match_socket.buf_outsize2=match_socket.buf_outsize3;

 match_socket.buf_outsize3=0;

}


// Frame does not contain a TFIN so simply acknowledge it, if data od SYN received
if(!(hframe.tcp_hdr.flags&TFIN)){
 if(dlen || (hframe.tcp_hdr.flags&TSYN)) send_match_ok_TCP(0,0,TACK); // Frame OK: Acknowledge immediatelly if data...
}else if(hframe.tcp_hdr.flags&TFIN){ // Come to here if RST and/or received
 match_socket.sack.u++;  // Count remote FIN
 send_match_ok_TCP(0,0,TACK+TFIN+TPUSH); // Acknowledge + FIN
 match_socket.sseq.u++;  // Count our FIN after sending!...
 match_socket.state=TCP_FINCON; // FIN Confirmed
}

// Only if nothing available reset retry_counter...
if(!match_socket.buf_outsize1) match_socket.retry_cnt=0;
return EVENT_TCP_DATARECEIVED;

case TCP_FINSENT:
 //puts("<FINSENT>");
 if(!(hframe.tcp_hdr.flags&TACK)) break;
 // printf("Flags: %u\n",hframe.tcp_hdr.flags);
 // printf("<<M:%ld H:%lx >>",match_socket.sack.u,hframe.tcp_hdr.seq.u);
 if(match_socket.sack.u!=hframe.tcp_hdr.seq.u) return EVENT_TCP_OUTOFBOUNDS; // Ignore-out-of-bounds segments!
 //puts("<Wait3LASTACK>");
 if(hframe.tcp_hdr.flags&TFIN){    // Fin accepted by Remote!
  match_socket.sack.u++;    // Count remote FIN
  send_match_ok_TCP(0,0,TACK);   // Frame OK: Acknowledge immediatelly!
  free_match_socket(); // Free Buffers if allocated...
  match_socket.state=TCP_CLOSED;  // Connection ends NOW
  //puts("<FINAL ACK SENT CLOSED>");
 }
 match_socket.retry_cnt=0;
 return EVENT_TCP_WAITLASTACK;

case TCP_FINCON: // Accept one last ACK
 if(!(hframe.tcp_hdr.flags&TACK)) break;
 if(match_socket.sack.u!=hframe.tcp_hdr.seq.u) return EVENT_TCP_OUTOFBOUNDS; // Ignore-out-of-bounds segments!

 match_socket.state=TCP_CLOSED;  // Connection ends NOW
 free_match_socket(); // Free Buffers if allocated...
 // puts("<LAST FIN ACKNOWLEDGED>");
 match_socket.retry_cnt=0;
 return EVENT_TCP_CLOSED;
}

free_match_socket(); // Free Buffers if allocated...
match_socket.state=TCP_CLOSED;
send_incomming_reset_TCP(dlen,&remote_mac,remote_ip.ipl);  // Denie further request!
return EVENT_TCP_ILLEGALFRAME; // Denie illegal frames;
}

/*****************************************************************************
* uint final_timeout_socket();
```

```
*
* Socket has definitely timed out. Free it for other users...
*****************************************************************************/
uint final_timeout_socket(void){

 //printf("TIMEOUT match_socket_type: %u\n",match_socket.socket_type);
 if(match_socket.socket_type==SOCKET_TCP){
#ifdef USE_TCP_CLIENT
    if(match_socket.state<ARPSENT)
#endif
  send_match_ok_TCP(0,0,TRST); // Reset this socket!
  free_match_socket(); // Free Buffers if allocated...
  //puts("<TIMEOUT RESET>");
  match_socket.state=TCP_CLOSED;  // ==0 (for UDP as well)
  return EVENT_TCP_TIMEOUT;
 }
 match_socket.state=0;  // ==0 (for UDP as well, but no action required)
 return EVENT_SOCKET_TIMEOUT;
}

/*****************************************************************************
* uint retransmit_socket();
*
* Socket requires a retransmition
*****************************************************************************/
uint retransmit_socket(void){
 unsigned long hseq;

 // printf(" >>--RE-TX:%u>>",match_socket.sremote_port);

 if(match_socket.socket_type==SOCKET_TCP){
  switch(match_socket.state){
  case TCP_SYNCON:    // Timeout after SYN-confirmed->Transmit Confirmation again
   send_match_ok_TCP(0,0,TSYN+TACK); // Transmit Again
   // puts("<TCP RETRANSMIT TSYN+TACK>");
   return EVENT_TCP_RETRANS;

  case TCP_EST:     // Timeout in an established Connection
   if(match_socket.buf_outsize1){  // Something un-acknowledged?
    //puts("<TCP RETRANSMIT EST>");
    // Seq. represents the sent data, so for resend subtract the block from seq, afterwards ad it...
    hseq=match_socket.sseq.u; // Save current Sequ (Pos.)
    // Rewind to Pos. before BUF1 was sent

    match_socket.sseq.u=match_socket.sseq_1-match_socket.buf_outsize1; // 32 Bit operation - This must be acknowledged to
free the buffer.
    send_match_ok_TCP(match_socket.p_outbuf1,match_socket.buf_outsize1,TACK+TPUSH);
    match_socket.sseq.u=hseq;  // Restore old Pointer

    return EVENT_TCP_RETRANS;
   }
   // puts("<TCP RT IDLE IDLE>");
   // Stack is idle: All ok
   match_socket.timer=TCP_IDLE_RETRIES;  // Socket OK, LONG TIMEOUT!!!
   return 0;

  case TCP_FINCON:
  case TCP_FINSENT:
   send_match_ok_TCP(0,0,TFIN+TACK+TPUSH); // Transmit, without any data after FIN_CON...
   // puts("<TCP FIN RETRANSMIT>");
   return EVENT_TCP_RETRANS;

#ifdef USE_TCP_CLIENT
  case ARPSENT:
   send_request_ARP(match_socket.sremote_ip);
   // puts("<(TCP) ARP RETRANSMIT>");
   return EVENT_TCP_RETRANS;

  case ARPREC:
   // puts("<(TCP) ARP-REQUEST RECEIVED!!!>");

   // Ports already setup!
   match_socket.sseq.w.h_word=net_service_cnt;   // Time ascending...
   match_socket.sseq.w.l_word=0;    // Our relative Pointer (for HTTP, -1 due to SYNC)
   match_socket.state=TCP_SYNSENT;  // SYN confirmed with SYN+ACK
```

```
 case TCP_SYNSENT:
  send_match_ok_TCP(0,0,TSYN); // Initiate Connection with a SYN
  // puts("<ACTIVE SYN SENT>");
  return 0; // Only 1 Try, No Retransmition!
#endif
  }
 }

#ifdef USE_UDP_CLIENT
    else if(match_socket.socket_type==SOCKET_UDP){
       switch(match_socket.state){
       case ARPSENT:
          send_request_ARP(match_socket.sremote_ip);
          // puts("<(UDP) ARP RETRANSMIT>");
          return EVENT_UDP_ARPRETRANS;

       default:
          // puts("<(UDP) TIMEOUT with ARP-REQUEST RECEIVED!!!>");

          match_socket.retry_cnt=0;                // Never close an ARPED UDP-Socket...
          match_socket.timer=UDP_IDLE_RETRIES;     // Socket OK, LONG TIMEOUT!!! No change in state
          return 0;
       }
   }
#endif


 return EVENT_SOCKET_RETRANS;
}

/*****************************************************************************
* uint periodical_socket();
*
* Watch non-0-state sockets periodically every 0.5 secs...
*****************************************************************************/
uint periodical_socket(void){
 uchar h;
 // First decrement sub-timer. If no 0: No Action required

 h=match_socket.timer-1;
 if(h){
  match_socket.timer=h;
  return 0;
 }


 match_socket.timer=BASIC_RETRY_TIMER;
 h=match_socket.retry_cnt+1;
 if(h==MAX_RETRIES){
  return final_timeout_socket();
 }else{
  match_socket.retry_cnt=h; // Retry again...
  return retransmit_socket();
 }
}

/*****************************************************************************
* void process_TCP(void)
*
* 3.rd-Level-Multiplexer
* A note for reading UDP-Datagrams: if Size is odd, last byte is in the
* HBYTE of the last Read_Frame_word_8900()...
* Usually a TCP-frame will never come as broadcast, so treat each as more
* important than other types...
*****************************************************************************/
uint process_TCP(uint dlen){
    xdata UC_SOCKET *psock;
    uchar ui;
 uchar ohlen;
    uint res;

 Read_Frame_xdata_8900(hframe.bytes,20); // Read informative part of TCP header to HFRAME
```

```
 dlen-=20;    //

 ohlen=hframe.tcp_hdr.hlen-80;
 while(ohlen){  // Eat TCP-option, if MSS: ignore silently...
  ohlen-=16; // ohlen = size in 32-bit-word<<4
  dlen-=4;
  Read_Frame_long_8900();
 }

#ifdef DEBUG_REC
    // Record Data of received Frame
    record_frame('R',hframe.tcp_hdr.sport,hframe.tcp_hdr.seq.u,hframe.tcp_hdr.ack.u, hframe.tcp_hdr.flags,dlen);
#endif

 // First try: Find any MATCHING socket. If one found, copy and process it...
     // This will also find a closed socket for a ACK-FIN->ACK-retransmition...
     psock=uc_socket;
     for(ui=0;ui<MAX_SOCK;ui++,psock++){
        if(psock->socket_type==SOCKET_TCP){  // Only TCP-Sockets are of interest
            // Test Remote IP-Match-Match,remote port and local port
            if(psock->sremote_ip==remote_ip.ipl){
               if(psock->sremote_port==hframe.tcp_hdr.sport){
                  if(psock->local_port==hframe.tcp_hdr.dport){
        // First copy to MATCH_SOCKET
      xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
                     res=state_machine_TCP(dlen);     // Now Header read, ready to read data
        // Copy back from MATCH_SOCKET and return
      xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
                     return res+ui;
                    }
               }
            }
        }
     }

     // Now: No matching Socket found: Then only frames with SYN are allowed!
     if(!(hframe.tcp_hdr.flags&TSYN)) return EVENT_TCP_ILLEGALFRAME;
       // No matching socket has been found, so find one with TCP_CLOSED and matching local port to open as a new one...
     psock=uc_socket;
     for(ui=0;ui<MAX_SOCK;ui++,psock++){
        if(psock->socket_type==SOCKET_TCP){  // Only TCP-Sockets are of interest if an offered local port is matched
            if(psock->state==TCP_CLOSED && psock->local_port==hframe.tcp_hdr.dport){
    // First copy to MATCH_SOCKET
    xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
               res=state_machine_TCP(dlen);     // Now Header read, ready to read data
        // Copy back from MATCH_SOCKET and return
    xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
               if (res) return res+ui;
            }
        }
     }

     // Nothing found and nothing free! Deny request by replying with a TCP-RESET (not replying may be safer but unpolite...)
     send_incomming_reset_TCP(dlen,&remote_mac,remote_ip.ipl);
     return EVENT_TCP_DENIED;

}



/*****************************************************************************
* void process_IP(void)
*
*
* 2.nd-Level-Multiplexer
*****************************************************************************/
uint process_IP(void){
 uint hdr;
 uint dlen;
 uchar pcol;

 hdr=Read_Frame_word_8900();  // Read Header
 if((hdr&0xF000)!=0x4000) return EVENT_IP_NOIP4; // Not IP4!
 dlen=Read_Frame_word_8900();  // Read total length of datagram
 Read_Frame_word_8900();   // Ignore Ident
```

```
 if(Read_Frame_word_8900()&0x3FFF) return EVENT_IP_WONTFRAG; // Reject fragemnts!

 pcol=(uchar)Read_Frame_word_8900(); // Protocol (1: ICMP 6 TCP 17: UDP)
 Read_Frame_word_8900();   // Ignore IP Checksum (already secured by Ethernet)

 remote_ip.ipl=Read_Frame_long_8900(); // Destination IP (should be US)
 Read_Frame_long_8900();   // Destination IP (should be US)

 dlen-=20;    // Adjust header
 hdr&=0xF00;
 hdr>>=8;
 hdr-=5;
 while(hdr--){
  Read_Frame_long_8900();  // Ignore IP options
  dlen-=4;
 }
 if(pcol==1){
  return process_ICMP(dlen);
 }else if(pcol==6){ // TCP
  return process_TCP(dlen);
#ifdef USE_UDP
 }else if(pcol==17){ // UDP
  return process_UDP(dlen);
#endif
 }
 return EVENT_IP_UNKNOWN;  // Don't unterstand this
}


#ifdef USE_UDP
/*****************************************************************************
* uint send_socket_udp(uchar sock, xdata uchar* pdata, uint datalen)
*
* Send data without any buffering
*****************************************************************************/
uint send_socket_udp(uchar sock, xdata uchar* pbuf, uint datalen){
    xdata UC_SOCKET *psock;
    psock=&uc_socket[sock];
 if(psock->socket_type!=SOCKET_UDP) return EVENT_UDP_ERROR;
 send_upd(pbuf,datalen,psock->sremote_mac,psock->sremote_ip,psock->local_port,psock->sremote_port);
 return 0;
}
#endif

/*****************************************************************************
* uint send_socket_tcp(uchar sock, xdata uchar* pdata, uint datalen)
*
* Bind an (allocated and filled xdata) buffer to a socket and send it. After Success,
* the buffer is freed by the stack (check with ready4tx_socket()
* The buffer must be allocated with allocate_tx_buf().
* For return values!=0 the buffer must be freed by the caller!
*
*****************************************************************************/
uint send_socket_tcp(uchar sock, xdata uchar* pbuf, uint datalen){
    xdata UC_SOCKET *psock;
    psock=&uc_socket[sock];
 // Copy Socket to Working Socket
 xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
 if(match_socket.socket_type!=SOCKET_TCP || match_socket.state!=TCP_EST) return EVENT_TCP_DENIED;
 if(!datalen) {
  free_tx_buf(pbuf); // Free Buffer
  return 0;  // IDIOTA!
 }

 // Bind Buffer try to allocate B1 first, then B2m then B3 else error
 if(!match_socket.buf_outsize1){
  match_socket.p_outbuf1=pbuf;
  match_socket.buf_outsize1=datalen;
  match_socket.sseq_1=match_socket.sseq.u+datalen;
  // puts("<SB1>");
 }else if(!match_socket.buf_outsize2){
  match_socket.p_outbuf2=pbuf;
  match_socket.buf_outsize2=datalen;
  match_socket.sseq_2=match_socket.sseq.u+datalen;
```

```
 // puts("<SB2>");
 }else if(!match_socket.buf_outsize3){
  match_socket.p_outbuf3=pbuf;
  match_socket.buf_outsize3=datalen;
  match_socket.sseq_3=match_socket.sseq.u+datalen;
  // puts("<SB3>");
 }else{
  // If data still pending: Error, Important: BUFFER NOT FREED!
  return EVENT_TCP_TXPENDING; // Can't send, old data still waiting...
 }
 send_match_ok_TCP(pbuf,datalen,TACK+TPUSH);
 match_socket.sseq.u+=datalen; // 32 Bit operation - This must be acknowledged to free the buffer.

 // New TIMEOUT
 match_socket.retry_cnt=0;
 match_socket.timer=BASIC_RETRY_TIMER;

  // Copy back from MATCH_SOCKET and return
 xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
 return 0; // All OK
}

/*****************************************************************************
* uint notready4tx_socket_tcp(uchar sock)
*
* Querries if a TCP socket is ready for Transmition, ok if 0.
* Checks if a Buffer is available for transmition to!)
*
* Flag: RDY_4_TX (>0) or RDY_4_CLOSE (0)
*****************************************************************************/
uint notready_socket_tcp(uchar sock, uchar flag){
    xdata UC_SOCKET *psock;
    psock=&uc_socket[sock];
 if(psock->socket_type!=SOCKET_TCP || psock->state!=TCP_EST) return EVENT_TCP_DENIED;
 if(!tx_bufleft) return EVENT_SOCKET_NOBUFFER;  // Stack may be ready, but no buffer available...

 if(flag){ // Check Ready for TX: BUF3 must be empty
  // If data still pending (Output Buffer full): Error
  if(psock->buf_outsize3) return EVENT_TCP_TXPENDING; // Can't send, old data still pending
 }else{  // Check Read for Close: BUF1 must be empty
  if(psock->buf_outsize1) return EVENT_TCP_TXPENDING; // Can't send, old data still pending

 }
 return 0; // SOCKEt IS READY!
}


/*****************************************************************************
* uint stringsend_socket_tcp(uchar sock, far char* pdata);
*
* Allocate a TCP-TX-Buffer and copy a string (far!) into it.
* Returns 0 on success. Calls send_socket_tcp().
*****************************************************************************/
uint stringsend_socket_tcp(uchar sock, far char* pdata){
 xdata uchar* pbuf;
 uint datalen;

 // Check if allowed
 if(notready_socket_tcp(sock,RDY_4_TX)) return EVENT_TCP_DENIED;
 datalen=strlen(pdata);
 if(datalen>MAX_TX) return EVENT_SOCKET_BUF2SMALL; // Can't send as much...
 // Allocate a buffer
 pbuf=allocate_tx_buf();
 if(!pbuf) return EVENT_SOCKET_NOBUFFER;   // No Buffer free?? -> Memory corrupt!
 bmove(pdata,pbuf,datalen);
 return send_socket_tcp(sock,pbuf,datalen);
}


/*****************************************************************************
* uint close_socket_tcp(sock)
*
* Close an open socket (regular mode)
*
*****************************************************************************/
```

```
uint close_socket_tcp(uchar sock){
    xdata UC_SOCKET *psock;
    psock=&uc_socket[sock];
 // Copy Socket to Working Socket
 xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
 if(match_socket.socket_type!=SOCKET_TCP || !match_socket.state) return EVENT_TCP_DENIED; // Closing always
possible...

 // If data still pending: Error
 if(match_socket.buf_outsize1) return EVENT_TCP_TXPENDING; // Can't send, old data still waiting...

 send_match_ok_TCP(0,0,TACK+TFIN+TPUSH);
 match_socket.sseq.u++; // 32 Bit operation - This must be acknowledged
 match_socket.state=TCP_FINSENT;

 // New TIMEOUT
 match_socket.retry_cnt=0;
 match_socket.timer=BASIC_RETRY_TIMER;

  // Copy back from MATCH_SOCKET and return
 xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));

 // printf("<--CLOSE %u-->",match_socket.sremote_port);

 return 0; // All OK
}

#ifdef USE_TCP_CLIENT
/*****************************************************************************
* uint open_socket_tcp(sock,ipl,port);
*
* Initiate an active Open for a  given Socket
*****************************************************************************/
uint open_socket_tcp(uchar sock,unsigned long remote_ipl,unsigned int remote_port){
    xdata UC_SOCKET *psock;
    psock=&uc_socket[sock];
 // Copy Socket to Working Socket
 xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
 if(match_socket.socket_type!=SOCKET_TCP || match_socket.state) return EVENT_TCP_DENIED; // No Access to non

 match_socket.sremote_ip=remote_ipl;
 match_socket.sremote_port=remote_port;

 send_request_ARP(remote_ipl);
 match_socket.state=ARPSENT;

 // New TIMEOUT
 match_socket.retry_cnt=0;
 match_socket.timer=BASIC_RETRY_TIMER;

  // Copy back from MATCH_SOCKET and return
 xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
 return 0; // All OK
}
#endif

#ifdef USE_UDP_CLIENT
/*****************************************************************************
* uint open_socket_udp(sock,ipl,port);
*
* Initiate an active Open for a  given Socket in UDP-Mode
*
*****************************************************************************/
uint open_socket_udp(uchar sock,unsigned long remote_ipl,unsigned int remote_port){
      xdata UC_SOCKET *psock;
      psock=&uc_socket[sock];
   // Copy Socket to Working Socket
   xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
   if(match_socket.socket_type!=SOCKET_UDP || match_socket.state) return EVENT_UDP_DENIED; // No Access to non

   match_socket.sremote_ip=remote_ipl;
   match_socket.sremote_port=remote_port;

   send_request_ARP(remote_ipl);
   match_socket.state=ARPSENT;
```

```
   // New TIMEOUT
   match_socket.retry_cnt=0;
   match_socket.timer=BASIC_RETRY_TIMER;

   // Copy back from MATCH_SOCKET and return
   xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
   return 0; // All OK
}
#endif

#ifdef USE_UDP
/*****************************************************************************
* uint close_socket_udp(sock)
*
* Close an open socket (regular mode)
*
*****************************************************************************/
uint close_socket_udp(uchar sock){
       xdata UC_SOCKET *psock;
       psock=&uc_socket[sock];
   // Copy Socket to Working Socket
   xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
   if(match_socket.socket_type!=SOCKET_UDP || !match_socket.state) return EVENT_UDP_DENIED; // Closing always
possible...
   match_socket.state=0;   // That's all to close...
   // Copy back from MATCH_SOCKET and return
   xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
   return 0; // All OK
}
#endif



/*****************************************************************************
* uint poll_net(void)
*
* Top-Level-Multiplexer, should be happy with SNAP frames too...
* Will return !=0 if Event was encountered
*****************************************************************************/
uint poll_net(void){
 uint RxEvent;
 uint type;

 RxEvent=Read_PP_8900(PP_RxEvent);
 if(RxEvent & RX_OK) {
  Read_FrameHL_word_8900(); // Skip Status HL
  Read_FrameHL_word_8900(); // Read Length HL (delivered >= 60!)
  Read_Frame_word_8900(); // Skip OUR MAC... (6 Bytes)
  Read_Frame_long_8900();
  Read_Frame_xdata_8900(&remote_mac[0],6); // Read Sender's MAC
  type=Read_Frame_word_8900();
  if(type<=0x5DC){ // SNAP Frame! Eat LSAP-Ctrl-OUI and retry...
   if(net_match_uint(0xAAAA)) return 0;
   if(net_match_ulong(0x3000000)) return 0;
   type=Read_Frame_word_8900(); // Read NEW type...
  }
  // *** First stage input filter/multiplexer for received frames ***
  if(type==0x0806){ // This is an ARP-Frame!
   return process_ARP();
  }else if(type==0x800){ // IP Header!
   return process_IP();
              } // ignore unknown frames
 }else{
  /*** Do soemthing periodically net_timer decremented 2 times per Sec!... ***/
  if(!net_timer){
      xdata UC_SOCKET *psock;
      uchar ui;
   uint res;
   net_service_cnt++;  // Sequence-Timer Highbyte
   net_timer=TIMER_FRQ/2; // about 0.5 Hz ONLY after one complete IDLE-pass...
      psock=uc_socket;
      for(ui=0;ui<MAX_SOCK;ui++,psock++){
         if(psock->state){  // Examine only non-0-state-Sockets
       // First copy to MATCH_SOCKET
```

```
    xram_fast_copy((xdata uchar*)psock,(xdata uchar*)&match_socket,sizeof(UC_SOCKET));
                    res=periodical_socket();     // Retry transmition... (Could be UDP for ARP as well...)
       // Copy back from MATCH_SOCKET and return
    xram_fast_copy((xdata uchar*)&match_socket,(xdata uchar*)psock,sizeof(UC_SOCKET));
                    if(res) return res+ui; // Return immediatelly if necessary
   }
  }
  return EVENT_SOCKET_IDLETIMER; // About twice/sec
 }

 }
 return 0; // NO EVENT
}



/*****************************************************************************
* IRQ: The system timer. Count down net_timer, leave it if 0!
*****************************************************************************/
#pragma option -g0  // We don't want to have debug info in the interrupt
IRQ_VECTOR(timer0int,TIMER0)
void timer0int (void) interrupt  {
 uchar h;
 h=net_timer;
 if(h) net_timer=(--h);
}
#pragma option -g   // Restore debug info level to default

/*****************************************************************************
* uchar Init_net()
*
* Initialise Network, return 0 if OK, else ERROR
*****************************************************************************/
uchar Init_net(void){


 my_mac[0]=0; // High 4 of MAC Bytes fixed to 0
 my_mac[1]=0x51;

 xram_fast_copy(my_ip.bytes,my_mac+2,4); // Lower 4 Bytes: IP of this node


 if(Init_8900()) return 1; // ERROR (MAC set as global!)

 // Use Timer 0 as TCP-Timer
 EA=0;  // Disable all IRQ

 TMOD &=0xF0;
 TMOD |=0x01; // 16 bit Prescaler: 28.125 Hz
 TR0=1;  // Timer 0 RUN
 ET0=1;

 EA=1;  // Enable all IRQ
       _wait_ms(100); // May needs a few msec until ready
 return 0;
}


// END
```

✓ *Net.h*

```
#define MAX_RX 100   // May be up to 1400-1500 Bytes, use at least 100 Bytes
// TX-Buffers (>=1)
#define MAX_TX 100  // Transmition Buffer Size, may be up to 1400-1500 Bytes

#define MAX_SOCK 5  // Maximum No. of Sockets (1-8 recommended)
#define TX_BUFFERS 4  // Recommended: 3 to 3*MAX_SOCK



/*****************************************************************************
* Internal Options (Changes not recommended)
*****************************************************************************/
```

```
// *** Timeouts-Definitions ***
#define MAX_RETRIES 4 // No of Retries (ARP/UDP/TCP...)
#define BASIC_RETRY_TIMER 6 // in 0.5 Seconds (WIN32 uses 3 sec?)
#define TCP_IDLE_RETRIES 40 // == TOTAl TIMEOUT=((TCP_IDLE_RETRIES*(MAX_RETRIES-
1)+BASIC_RETRY_TIMER)/2 ins sec, (40 ca. 1 min )
#define UDP_IDLE_RETRIES 40 // == TOTAl TIMEOUT=((UDP_IDLE_RETRIES*(MAX_RETRIES-
1)+BASIC_RETRY_TIMER)/2 ins sec, (40 ca. 1 min )


/* tcp_client_flag */
#define FLAG_ACTIVE_OPEN 1 // Only used for TCP_CLIENTS, else ignored (= Passive open only poss.)
#define FLAG_PASSIVE_OPEN 0 // dto.

/* Socket type */
#define SOCKET_NONE 0 // Not active
#ifdef USE_UDP
 #define SOCKET_UDP 1 // UDP-Socket
#endif
#define SOCKET_TCP 2 // TCP_IP-Socket

/* Public Socket states */
#define TCP_CLOSED  0 // 0 for all: Socket closed (and listen)
#define TCP_EST  2    // Established, Connection OK

#ifdef USE_UDP_CLIENT
 #define UDP_EST  7  // For UDP Established is the same as ARP Received...
#endif



/* Buffer-Flags for notready_socket_tcp() */
#define RDY_4_TX 1
#define RDY_4_CLOSE 0


/*****************************************************************************
* Global structures and data
*****************************************************************************/
typedef union{
 uchar bytes[4];  // Byte representation
 unsigned long ipl; // 32bit representation
} IP_ADR;

typedef union{
 unsigned long u;
 struct{
  uint h_word;
  uint l_word;
 } w;
} WORD2_LONG;

typedef uchar  MAC[6]; // define it as a type

typedef struct{
 uchar  socket_type;  // Usage for this socket, see above

 uchar state;   // TCP-Socket state machine (TCP_CLOSED=0....)

#ifdef USE_TCP_CLIENT
 uchar  tcp_client_flag; // Only required for TCP clients to dist. between active/passive open.
#endif
 MAC sremote_mac;  // Remote's node MAC
 unsigned long sremote_ip; // Remote node's IP
 uint sremote_port;  // Remote's port
 uint local_port;  // Our port for UDP and TCP (HTTP==80)

 WORD2_LONG sack;  // TCP-Acknowledge nr. (of remote node)
 WORD2_LONG sseq;  // TCP-Sequence nr. (== rel. Data Pointer)

 uchar retry_cnt;  // Up-Counter for retries (may be later used as an exponent for timer)
 uchar timer;   // Counts down to 0, then something may happen or not...

 // Here 3 Packets for each Socket reserved
```

```
 uint  buf_outsize3;  // 0 for NO-Data, Size o following pointer...
 unsigned long sseq_3;  // TCP-Sequence nr. (== rel. Data Pointer for this pack)
    xdata uchar *p_outbuf3;       // Pointer to avtive fragment 3 for this socket (outgoing)

 uint  buf_outsize2;  // 0 for NO-Data, Size o following pointer...
 unsigned long sseq_2;  // TCP-Sequence nr. (== rel. Data Pointer)
    xdata uchar *p_outbuf2;       // Pointer to avtive fragment 2 for this socket (outgoing)

 uint  buf_outsize1;  // 0 for NO-Data, Size o following pointer...
 unsigned long sseq_1;  // TCP-Sequence nr. (== rel. Data Pointer)
    xdata uchar *p_outbuf1;       // Pointer to avtive fragment 1 for this socket (outgoing)

} UC_SOCKET;


extern MAC my_mac;  // init to ($51:$80:IP)

extern IP_ADR my_ip;  //  init manually to desired IP for this machine (public)
extern IP_ADR remote_ip; // Last read IP (can be used as a temporary (see demos)

#if defined(USE_TCP_CLIENT) || defined(USE_UDP_CLIENT)
 extern IP_ADR subnet_ip;  // init to subnetmask (usually: 255,255,255,0}, Subnet mask for this node
 extern IP_ADR gateway_ip;   // init to  Gateway-IP for this node
#endif

// *** THE SOCKETS ***
extern UC_SOCKET uc_socket[MAX_SOCK];  // My (User's) Sockets!


/*****************************************************************************
* The public buffers in XRAM
*****************************************************************************/

// RX-Buffer (1)
extern uchar rcv_buf[MAX_RX];   // Buffer for receiving data (Mainly HTTP-Header...)
extern uint rcv_len;    // Size of received data (int)


/*****************************************************************************
* Useful Macros
*****************************************************************************/
// A Macro for composing IP-Adrs. (Usage: COMPOSE_IP(remote_ip,192,168,1,5);)
#define COMPOSE_IP(x,a,b,c,d)  {x.bytes[0]=a;x.bytes[1]=b;x.bytes[2]=c;x.bytes[3]=d;}

// A Macro for setting up an socket. tcp_socket_flags only used for clients...
#ifdef USE_TCP_CLIENT
 #define SOCKET_SETUP(a,b,c,d) {uc_socket[a].socket_type=b; uc_socket[a].local_port=c; uc_socket[a].tcp_client_flag=d;}
#else
 #define SOCKET_SETUP(a,b,c,d) {uc_socket[a].socket_type=b; uc_socket[a].local_port=c;} // Par. d only required if
TCP_CLIENT, ignored!
#endif


/*****************************************************************************
* NETWORK Events
*****************************************************************************/
// ========GROUP 9: <HTTP>=======================
// --- HTTP EVENTS ---
#define EVENT_HTTP_REQUEST 0x9000  // Somebody has requested a Page


// ========GROUP A: <ARP>========================
// --- ARP EVENTS ---
#define EVENT_ARP_REQUEST 0xA000  // Somebody requestet OUR IP (no important)
#define EVENT_ARP_OTHERREPLY 0xA100  // Received a Reply to (our???) ARP Request?
#define EVENT_ARP_UNKNOWN 0xA200  // Unknown ARP frame received
#define EVENT_ARP_NOTYPE 0xA300  // ARP with unknown Type received
#define EVENT_ARP_OTHER  0xA400  // Arp's MAC did'nt match our
#define EVENT_ARP_OURREPLY 0xA500  // ARP received as a Reply to OUR Request

// ========GROUP B: <ICMP>=======================
// --- ICMP EVENTS ---
#define EVENT_ICMP_REPLY 0xB000  // Received a reply to a PING?
#define EVENT_ICMP_REQUEST 0xB100  // We have been PINGed! (This EVENT could be of interest...)
#define EVENT_ICMP_UNKNOWN 0xB200  // Unknown ICMP frame received
```

```
// ========GROUP C: <IP>=======================
// --- IP EVENTS ---
#define EVENT_IP_NOIP4  0xC000  // Accept only IP4 Frames
#define EVENT_IP_WONTFRAG 0xC100  // We don't accept fragemnts!
#define EVENT_IP_UNKNOWN 0xC200  // received unknown IP

// ========GROUP D: <Gen. sockets>==============
// --- General socket EVENTS ---
#define EVENT_SOCKET_TIMEOUT 0xD000  // General non-TCP-Socket-TIMEOUT (for TCP: _TCP_TIMOUT)
#define EVENT_SOCKET_RETRANS 0xD100  // General non-TCP-Socket-Retransmition
#define EVENT_SOCKET_NOBUFFER 0xD200  // No Buffer available
#define EVENT_SOCKET_BUF2SMALL 0xD300  // Buffer is too small
#define EVENT_SOCKET_IDLETIMER 0xD400  // Idle-Time triggered (about 2/sec)

// ========GROUP E: <UDP>======================
// --- UDP EVENTS ---
#define EVENT_UDP_UNSOLICITED 0xE000  // Received Unsolicited UDP-DATA...
#define EVENT_UDP_ERROR  0xE100  // UDP segment with error received (ignored)
#define EVENT_UDP_DATARECEIVED 0xE200  // Received UDP-Data, lower 3 Bits contain the socket no. Len separate...
#define EVENT_UDP_ARPRETRANS 0xE300  // UDP-ARP-Retransmition due to timeout
#define EVENT_UDP_DENIED 0xE400  // Denied an incomming or outgoing request (maybe socket failure or state error)!

// =======GROUP F: <TCP>========================
// --- Misc. TCP EVENTS  (less important) ---
#define EVENT_TCP_DENIED     0xF000      // Denied an incomming or outgoing request (maybe socket failure or state error)!
#define EVENT_TCP_RETRANS 0xF100  // TCP-Retransmition due to timeout
#define EVENT_TCP_SYNRECEIVED  0xF200  // <Passive Open>: SYN received, replied with an SYN+ACK. Stack-Idx in
LowByte
#define EVENT_TCP_OUTOFBOUNDS 0xF300  // Out-of-Bound-Segment received (Segment ignored)
#define EVENT_TCP_TXPENDING 0xF400  // Can not send, old data not acknowledged!

// --- Only ONE EVENT indicates received data (or ACK) on an (open) Socket ---
#define EVENT_TCP_DATARECEIVED 0xF500  // Received rcv_len data in rcv_buf[] (== Implicit Passive Open)

// --- These TCP EVENTS cause closing the socket (Bitmask 0xF800 or Cond. >= 0xF800)
#define EVENT_TCP_ILLEGALFRAME  0xF800    // Received a totally-out-or-order TCP frame
#define EVENT_TCP_RESETRECEIVED 0xF900  // Received a RESET-Signal for this Socket
#define EVENT_TCP_TIMEOUT 0xFA00  // Connection closed because of timeout
#define EVENT_TCP_WAITLASTACK 0xFB00  // Waiting for Last Acknoledge (only in active mode)
#define EVENT_TCP_CLOSED 0xFC00  // Closing Socket (regular)



/*****************************************************************************
* Functions (some are reserved for internal usage)
*****************************************************************************/

// Low Level (Use only in special cases).
xdata uchar* allocate_tx_buf(void);
void free_tx_buf(xdata uchar* pbuf);
void free_match_socket(void);
void send_request_ARP(unsigned long ipl);
void send_udp(xdata char* , uint len,xdata MAC *pmac,unsigned long rem_ipl,uint sport, uint dport);


// TCP
uint send_socket_tcp(uchar sock, xdata uchar* pbuf, uint datalen);
uint notready_socket_tcp(uchar sock, uchar flag);
uint stringsend_socket_tcp(uchar sock, far char* pdata);
uint close_socket_tcp(uchar sock);
uint open_socket_tcp(uchar sock,unsigned long remote_ipl,unsigned int remote_port);

// UDP
uint open_socket_udp(uchar sock,unsigned long remote_ipl,unsigned int remote_port);
uint send_socket_udp(uchar sock, xdata uchar* pbuf, uint datalen);
uint close_socket_udp(uchar sock);

// General
uint poll_net(void);
uchar Init_net(void);
```

```
#ifdef DEBUG_REC

// --- Debugging stuff (only available if DEBUG_REC defined )
#define MAX_REC_FRAME   100 // Max. No. of frames to record
extern uint rec_no;

void record_frame(uchar typ, uint port, unsigned long seq, unsigned long ack, uchar flags, uint len);
uchar show_frame(uint no);

#endif

/*****************************************************************************/


// EOF
```

✓ *cs8900.c*

```
#include <stdio.h>

#ifdef FLEXGATE
 #include <reg535.h>
#endif
#ifdef ELMET
 #include <reg51.h>
#endif

#include "net.h"  // MAC
#include "cs8900.h"  // CS8900 Register Definitions


#ifdef FLEXGATE
 #define CS8900_RESET  P5_B4 // Hardware Pin for RESET (FlexGate @ C515)
#endif
#ifdef ELMET
 #define CS8900_RESET  P1_B7 // Hardware Pin for RESET (ELEKTOR-FlexGate @ MSC1210)
#endif

/*****************************************************************************
* void Write_8900(uint Register, uint Data);
*
* writes a word in little-endian byte order to a specified register
*****************************************************************************/
#asm
 .segment __Write_8900
 .export _Write_8900
_Write_8900:
 mov DPH,R6
 mov DPL,R7
 mov A,R5
 movx @DPTR,A
 inc DPTR
 mov A,R4
 movx @DPTR,A
 ret
#endasm



/*****************************************************************************
* void Write_PP_8900(uint Register, uint Data);
*
* writes a word in little-endian byte order to a specified Packet Page register
*****************************************************************************/
#asm
 .segment __Write_PP_8900
 .export _Write_PP_8900
_Write_PP_8900:
 mov DPTR,#ADD_PORT ; Def: $000A
 mov A,R7
 movx @DPTR,A
 inc DPTR
```

```
 mov A,R6
 movx @DPTR,A
 mov DPTR,#DATA_PORT ; Def: $000C
 mov A,R5
 movx @DPTR,A
 inc DPTR
 mov A,R4
 movx @DPTR,A
 ret
#endasm

/*****************************************************************************
* void Write_Frame_word_8900(uint Data);
*
* writes a word in little-endian byte order to the frame register
*****************************************************************************/
#asm
 .segment __Write_Frame_word_8900
 .export _Write_Frame_word_8900
_Write_Frame_word_8900:
 mov DPTR,#TX_FRAME_PORT  ; Def: $0000
 mov A,R6
 movx @DPTR,A
 inc DPTR
 mov A,R7
 movx @DPTR,A
 ret
#endasm

/*****************************************************************************
* void Write_Frame_long_8900(unsigned long Data);
*
* writes a long in little-endian byte order to the frame register
*****************************************************************************/
#asm
 .segment __Write_Frame_long_8900
 .export _Write_Frame_long_8900
_Write_Frame_long_8900:
 mov DPTR,#TX_FRAME_PORT  ; Def: $0000
 mov A,R4
 movx @DPTR,A
 inc DPTR
 mov A,R5
 movx @DPTR,A
 mov DPTR,#TX_FRAME_PORT  ; Def: $0000
 mov A,R6
 movx @DPTR,A
 inc DPTR
 mov A,R7
 movx @DPTR,A
 ret
#endasm

/*****************************************************************************
* void Write_Frame_xdata_8900(xdata uchar* ps, uint len);
*
* Writes a number of bytes from XRAM to Frame Register Standard-Order (Network Order)
* Coded in Assembler for maximum speed...
* May write an odd number of bytes if this is the LAST block written...
*****************************************************************************/
#asm
 .segment __Write_Frame_xdata_8900
 .export _Write_Frame_xdata_8900
_Write_Frame_xdata_8900:
 ; R67: Destination pointer
 ; R45: Len in bytes
 ; R23 used for temporaries
 mov A,R4 ; first divide len by 2
 clr C
 rrc A
 mov R4,A
 mov A,R5
 rrc A  ; now C set if odd len...
 mov R5,A
 mov A,R4 ; omit 0 words len
```

```
 orl A,R5
 jz ?wfx
 mov A,R5 ; prepare to use 2 djnz
 jz ?wf1
 inc R4
?wf1: mov DPH,R6 ; R67 holds XRAM adr.
 mov DPL,R7
 movx A,@DPTR
 mov R2,A
 inc DPTR
 movx A,@DPTR
 mov R3,A
 inc DPTR
 mov R6,DPH ; Word now in R23
 mov R7,DPL

 mov DPTR,#TX_FRAME_PORT  ; Def: $0000
 mov A,R2
 movx @DPTR,A
 inc DPTR   ; Def: $0001
 mov A,R3
 movx @DPTR,A

 djnz R5,?wf1
 djnz R4,?wf1
?wfx: jnc ?wfy
 mov DPH,R6
 mov DPL,R7
 movx A,@DPTR   ; get last Byte
 mov DPTR,#RX_FRAME_PORT  ; Def: $0000
 movx @DPTR,A
?wfy: ret

#endasm


/*****************************************************************************
* uint Read_PP_8900(uint Register);
*
* reads a word from a specified Packet Page register
*****************************************************************************/
#asm
 .segment __Read_PP_8900
 .export _Read_PP_8900
_Read_PP_8900:
 mov DPTR,#ADD_PORT ; Def: $000A
 mov A,R7
 movx @DPTR,A
 inc DPTR
 mov A,R6
 movx @DPTR,A
 mov DPTR,#DATA_PORT ; Def: $000C
 movx A,@DPTR
 mov R7,A
 inc DPTR
 movx A,@DPTR
 mov R6,A
 ret
#endasm

/*****************************************************************************
* uint Read_FrameHL_word_8900(void);
*
* reads a word from the Frame Register High-Low-Order! Req. for Status and Length
*****************************************************************************/
#asm
 .segment __Read_FrameHL_word_8900
 .export _Read_FrameHL_word_8900
_Read_FrameHL_word_8900:
 mov DPTR,#RX_FRAME_PORT+1 ; Def: $0001 Hi
 movx A,@DPTR
 mov R6,A
 mov DPTR,#RX_FRAME_PORT  ; Def: $0000 Low
 movx A,@DPTR
 mov R7,A
```

```
 ret
#endasm

/*****************************************************************************
* uint Read_Frame_word_8900(void);
*
* reads a word from the Frame Register Standard-Order (Network Order)
*****************************************************************************/
#asm
 .segment __Read_Frame_word_8900
 .export _Read_Frame_word_8900
_Read_Frame_word_8900:
 mov DPTR,#RX_FRAME_PORT  ; Def: $0000
 movx A,@DPTR
 mov R6,A
 inc DPTR
 movx A,@DPTR
 mov R7,A
 ret
#endasm


/*****************************************************************************
* unsigned long Read_Frame_long_8900(void);
*
* reads a long word from the Frame Register Standard-Order (Network Order)
*****************************************************************************/
#asm
 .segment __Read_Frame_long_8900
 .export _Read_Frame_long_8900
_Read_Frame_long_8900:
 mov DPTR,#RX_FRAME_PORT  ; Def: $0000
 movx A,@DPTR
 mov R4,A
 inc DPTR
 movx A,@DPTR
 mov R5,A
 mov DPTR,#RX_FRAME_PORT  ; Def: $0000
 movx A,@DPTR
 mov R6,A
 inc DPTR
 movx A,@DPTR
 mov R7,A
 ret
#endasm

/*****************************************************************************
* void Read_Frame_xdata_8900(xdata uchar* ps, uint len);
*
* read a number of bytes to XRAM from Frame Register Standard-Order (Network Order)
* Coded in Assembler for maximum speed...
* May read an odd number of bytes if this is the LAST block read...
*****************************************************************************/
#asm
 .segment __Read_Frame_xdata_8900
 .export _Read_Frame_xdata_8900
_Read_Frame_xdata_8900:
 ; R67: Destination pointer
 ; R45: Len in bytes
 ; R23 used for temporaries
 mov A,R4 ; first divide len by 2
 clr C
 rrc A
 mov R4,A
 mov A,R5
 rrc A  ; now C set if odd len...
 mov R5,A
 mov A,R4 ; omit 0 words len
 orl A,R5
 jz ?rfx
 mov A,R5 ; prepare to use 2 djnz
 jz ?rf1
 inc R4
?rf1: mov DPTR,#RX_FRAME_PORT  ; Def: $0000
 movx A,@DPTR
```

```
 mov R2,A
 inc DPTR   ; Def: $0001
 movx A,@DPTR
 mov R3,A   ; Word now in R23

 mov DPH,R6   ; R67 holds XRAM adr.
 mov DPL,R7
 mov A,R2
 movx @DPTR,A
 inc DPTR
 mov A,R3
 movx @DPTR,A
 inc DPTR
 mov R6,DPH
 mov R7,DPL

 djnz R5,?rf1
 djnz R4,?rf1
?rfx: jnc ?rfy
 mov DPTR,#RX_FRAME_PORT  ; Def: $0000
 movx A,@DPTR
 mov DPH,R6
 mov DPL,R7
 movx @DPTR,A
?rfy: ret

#endasm



/*****************************************************************************
* uchar Init_8900(void);
*
* Note: my_mac[] must be set before!
* Init the CS8900A. If OK, return 0, else: ERROR
*****************************************************************************/

uchar Init_8900(void){
  uint ui,mac;
  xdata uchar *pmac;

#ifdef CS8900_RESET   // If a Reset-Pin is defined...
  CS8900_RESET=0;   // Reset-Pulse must be at least 400 nsec!
  _wait_ms(100);
#endif

  ui=Read_PP_8900(PP_ChipID);  // Read Manufacturer


  if(ui!=0x630E) return 255;  // Failed to read CS8900A's ID!

  Write_PP_8900(PP_SelfCTL, POWER_ON_RESET);    // Soft-Reset the Ethernet-Controller

  while (!(Read_PP_8900(PP_SelfST) & INIT_DONE));    // wait until chip-reset is done

  // Read individual MAC from Array, convert it to 3 uints and set...
  pmac=my_mac;
  for(ui=PP_IA;ui<PP_IA+6;ui+=2){
   mac=(*pmac++) | ((*pmac++)<<8);  // Swap bytes bec. CS8900 is LE-machine...
   Write_PP_8900(ui, mac);   // Write MAC
  }


  Write_PP_8900(PP_RxCTL, RX_OK_ACCEPT | RX_IA_ACCEPT | RX_BROADCAST_ACCEPT);
  Write_PP_8900(PP_TestCTL, FDX_8900);  // Full Duplex
  Write_PP_8900(PP_LineCTL, SERIAL_RX_ON | SERIAL_TX_ON); // Transciever ON

  return 0;
}


/*****************************************************************************
* void Led_8900(char ns);
*
* Allow access to CS8900 Link-LED: 0: OFF, 1: ON, else: LINK
```

```
* Additional Function (just for fun)
*****************************************************************************/
void Led_8900(char ns){
  if(ns==1)  Write_PP_8900(PP_SelfCTL, 0x5000); // Led ON
  else  if(!ns) Write_PP_8900(PP_SelfCTL, 0x1000);// Led OFF
  else Write_PP_8900(PP_SelfCTL, 0x0000);// Led function LINK
}



/*****************************************************************************
* void RequestSend_8900(uint FrameSize);
*
* Note: Frame-Size in BYTES
* Requests space in CS8900's on-chip memory for storing an outgoing frame
* Refer to Fig. 5.12 data sheet for the bidding process
* This function must only be called ONCE, Will stall in an endless loop without
* net. So a maximum delay was added of 256 loops.
*****************************************************************************/
void RequestSend_8900(uint FrameSize){
  uchar del;
  Write_8900(TX_CMD_PORT, TX_START_ALL_BYTES);
  Write_8900(TX_LEN_PORT, FrameSize);
  del=0;
  while (!(Read_PP_8900(PP_BusST) & READY_FOR_TX_NOW)){
 del++;
   if(!del) break;
 // puts("CS8900 Rdy4Tx");
  }
}


// END
```

✓ *Set.html*

```
<html><head><title> Micro Web Server - Dynamic Form</title><meta http-equiv="Content-Type" content="text/html;
charset=windows-1253"><style type="text/css">
<!--
body {
        background-color: #000000;
}
-->
</style></head>
<body text=#FFFFFF vlink=#FFFFFF link=#FFFFFF alink=#FFFFFF>
<table align="center">
<tr><td width="640">
<hr noshade size="3" color="#00AAFF">
 <hr noshade size="3" color="#00AAFF">
<p><b><h2><i>Micro Web Server</i></h2>
</b></p>
<p>Dynamic Form. Set the ElmFlex's Clock and the LEDs. An invalid Time will be ignored. The Hardware contains two LEDs
(3+4), which can be controlled by the user. Each Set receives an ID, which will be displayed as "ID" on several pages. So you
can see, if the current settings are yours.<br>(Current) ID:@id
</p><form action="reply.html" method="get">
<table width=300><tr><td>Clock:
<input type=text name=A1 maxlength=2 value=@hr size=2> :
<input type=text name=A2 maxlength=2 value=@min size=2> :
<input type=text name=A3 maxlength=2 value=@sec size=2>
</td><td align=right><input type=submit value="Set Clock" name=A9>
</td></tr></table></form>
<form action="reply.html" method="get">
<table width=300><tr>
<td><input type=hidden name=A4>
LED3: <input type=checkbox name=A5 checke@ls3 value="ON"></td>
<td>LED4: <input type=checkbox name=A6 checke@ls4 value="ON"></td>
<td align=right><input type=submit value="Set LEDs" name=A9>
</td></tr></table></form>
<b><a href="home.html">Exit and Home...</a></b>
<hr noshade size="3" color="#00AAFF">
</td></tr></table>
</body></html>
```

✓ *Reply.html*

```
<html><head><title> Web Server - Reply</title><meta http-equiv="Content-Type" content="text/html; charset=windows-
1253"><style type="text/css">
<!--
body {
        background-color: #000000;
}
-->
</style></head>
<body text=#FFFFFF vlink=#FFFFFF link=#FFFFFF alink=#FFFFFF>
<table align="center"><tr><td width="640">
<hr noshade size="3" color="#00AAFF">
 <hr noshade size="3" color="#00AAFF">
<p><b><h2><i>Micro Web Server</i></h2>
</b></p>
<p>Accepted. (New) ID:@id</p>
<a href="set.html">Back...</a><br>
<a href="t_disp.html">Online Data...</a><br>
<a href="home.html">Home...</a>
<hr noshade size="3" color="#00AAFF">
</td></tr></table>
</body></html>
```

✓ *T_disp.html*

```
<p>Temperature/Clock. This page will be updated every 3 seconds. If other users make changes, the ID will reflect this.
ID:@id</p><p><b>Temperature Sensor:</b>
<table bgcolor=#ffffff border=1 cellpadding=0 cellspacing=0 width=514>
<tr><td><table width=@t_wid border=0 cellpadding=0 cellspacing=0>
<tr><td bgcolor=#ff0000> <b>@t_deg</b></td></tr></table></td>
</tr></table>
<table border=0 width=514><tr>
<td width="20%">|10°C</td><td width="20%">|15°C</td><td width="20%">|20°C</td><td width="20%">|25°C</td>
<td width="20%">|30°C</td></tr></table><br>
</p><p><b>Time:</b>
<table bgcolor=#0000FF border=1 cellspacing=0 width=100>
<tr><td align=center>@ctime</td></tr>
</table>
</p>
<div align="right"><b><a href="home.html">Exit and Home...</a></b>
   <br>
   <b><i>Gizmo productions</i></b></div>
<hr noshade size="3" color="#00AAFF">
</td></tr></table></body></html>
```

✓ *Home.html*

```
<html><head><title>Mico Web Server</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1"><style type="text/css">
<!--
body {
        background-color: #000000
}
-->
</style></head>
<body text=#FFFFFF vlink=#FFFFFF link=#FFFFFF alink=#FFFFFF>
<table width="648" height="400" align="center">
<tr><td width="640">
<hr noshade size="3" color="#00AAFF">
<div align="center">
  </div>
<hr noshade size="3" color="#00AAFF"><b>
<h2><i>Micro Web Server</i></h2>
</b></p>
```

```
<p>This is an Embedded Webserver based on the MSC1210-Board with an Ethernet-Extension-Board. The Software was written
with the uC/51 C Compiler. All source codes (a complete TCP/IP Stack!) are included in uC/51!!!.</p>
<table border=0>
<tr><td><b><a href="t_disp.html">Online Data</a></b></td><td>Temperature/Clock Display (Dynamic HTML)</td></tr>
<tr><td><b><a href="set.html">Setup</a></td></b><td>Set Clock and LEDs, get new ID (Dynamic Forms)</td></tr>
</table><br><hr noshade size="3" color="#00AAFF"><table width="640" border="0"><tr>
<td>ID:@id</td>
<td align="right"><div align="center">Hits:@hits (this page since Power On)<br>
   Gizmo production </div></td>
</tr></table></td></tr></table></body></html>
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Bert van Dam Μικροελεγκτες PIC. Elektor international media BV 2008

John Allwork NET Programming for Electronic Engineers C# 2008

The Microcontroller Idea Book Circuits, Programs, & Applications copyright 1994, 1997 by Jan Axelson ISBN 0-9650819-0-7 Published by Lakeview Research Distribution by International Thomson Publishing (ITP) in arrangement with Peer-to-Peer Communications

μC/51 V1.20.04 user's manual at www.wickenhaeuser.com

Cirrus Logic CS8900A Product Data Sheet, Crystal Lan Ethernet Controller

Hugo Cheung (Email: cheung_hugo@ti.com) Design Engineering Manager, High-Performance Analog, Data Acquisition Products MSC1210 debugging strategies for high-precision smart sensors.

www.elektor.com

www.el.teithe.gr