ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατασκευή Εφαρμογής Επικοινωνίας Συσκευών μέσω του Πρωτοκόλλου CAN (Controller Area Network)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Κ. Παπαναστασάτος

Επιβλέπων: Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατασκευή Εφαρμογής Επικοινωνίας Συσκευών μέσω του Πρωτοκόλλου CAN (Controller Area Network)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Κ. Παπαναστασάτος

Επιβλέπων: Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την …… 2008.

…… …… ……

Αθήνα, Οκτώβριος 2008

……
Αλέξανδρος Κ. Παπαναστασάτος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το αντικείμενο της διπλωματικής εργασίας ήταν η κατάσκευη μίας εφαρμογής βασισμένης στην επικοινωνία συσκευών μέσω του πρωτοκόλλου CAN (Controller Area Network). Κατασκευάστηκαν δύο τέτοιες συσκευές, στον πυρήνα των οποίων βρίσκεται ο μικροεπεξεργαστής LPC2129 της εταιρείας Philips Semiconductors που διαθέτει ενσωματωμένους ελεγκτές για το συγκεκριμένο πρωτόκολλο.

Για την επίδειξη της επικοινωνίας των δύο συσκευών, υλοποιήθηκε ένα σύστημα ανελκυστήρα που ελέγχεται τόσο τοπικά όσο και κεντρικά από προσωπικό υπολογιστή. Ο κεντρικός έλεγχος πραγματοποιείται διαμέσου μιας συσκευής που διασυνδέει το δίαυλο CAN με τον προσωπικό υπολογιστή μέσω σειριακής θύρας RS-232. Ο τοπικός έλεγχος πραγματοποιείται διαμέσου μιας δεύτερης συσκευής που είναι τοποθετημένη στο εσωτερικό της μινιατούρας ανελκυστήρα. Η συσκευή αυτή ελέγχει την κατάσταση των πλήκτρων ελέγχου, τους αισθητήρες και τις ενδείξεις του ανελκυστήρα, ενώ ταυτόχρονα συνδέεται στο δίαυλο CAN. Επιπλέον, παρέχει δύο σήματα για τον έλεγχο της κίνησης ενός βηματικού κινητήρα που μετακινεί τον ανελκυστήρα. Για την οδήγηση του βηματικού κινητήρα κατασκευάστηκε αντίστοιχος driver, συμβατός με τα σήματα που παράγει η δεύτερη συσκευή του δικτύου CAN.

Ο σκοπός της εργασίας ήταν η κατασκευή ενός ολοκληρωμένου συστήματος κατανεμημένου ελέγχου που προσομοιώνει σε μικρή κλίμακα πραγματικά συστήματα ελέγχου, καθώς επιτελεί τις συνήθεις και κύριες λειτουργίες που επιτελεί κάθε ελεγκτής, δηλαδή τον έλεγχο ψηφιακών εισόδων και εξόδων, τον έλεγχο κινητήρων μέσω κατάλληλων σημάτων και την επικοινωνία με το υπόλοιπο δίκτυο ελεγκτών και με τους χρήστες του συστήματος. Ο στόχος της ήταν η αντιμετώπιση των επιλογών, των αποφάσεων και των προβλημάτων που ανακύπτουν κατά τη σχεδίαση και υλοποίηση μίας εφαρμογής κατανεμημένου ελέγχου.

Λέξεις Κλειδιά

CAN, Controller Area Network, Σειριακή Επικοινωνία, Ανελκυστήρας, RS-232, Αυτόματος Έλεγχος, Κατανεμημένος Έλεγχος, Πρωτόκολλα Επικοινωνίας Ελεγκτών, Δίκτυα Ελεγκτών, Βηματικός Κινητήρας.

Abstract

The subject of this thesis was the construction of an application based on the communication of devices through the CAN (Controller Area Network) protocol. Two of these devices were developed, their core being the LPC2129 microprocessor available by Philips Semiconductors, which features embedded controllers for this specific protocol.

In order to exhibit the communication between these two devices, an elevator system controlled both locally and centrally by a personal computer was implemented. Central control is realized through a device interconnecting the CAN bus and the personal computer via a serial RS-232 port. Local control is realized through a second device placed inside the miniature elevator. This device checks the status of the control buttons, the sensors and the display of the elevator, while being connected to the CAN bus. Furthermore, it produces two signals for the motion control of a step motor which moves the elevator. In order to electrically drive the step motor, a driver compatible with the signals produced by the second device on the CAN network was constructed.

The purpose of this thesis was the construction of a complete distributed control system that simulates in small scale real control systems, as it carries out the main and most common operations of every controller, that is the control of digital inputs and outputs, the control of motors through suitable signals and the communication with the rest of the network of controllers as well as the users of the system. The goal was the confrontation with the choices, the decisions and the obstacles that arise while designing and building a distributed control application.

KeyWords

CAN, Controller Area Network, Serial Communication, Elevator, RS-232, Automatic Control, Distributed Control, Controller Communication Protocols, Controller Networks, Step Motor.

Ευχαριστώ θερμά:

τον κύριο Κιαμάλ Πεκμεστζή,
για την κατανόηση των υποκειμενικών και
αντικειμενικών δυσκολιών που αντιμετώπισα

το Χρήστο Τσακίρη,
για τη σημαντικότατη και απαραίτητη
ηθική και υλική υποστήριξη

το Μανώλη Τσακίρη,
για την επιστημονική και φιλοσοφική αρωγή

τη Νικολέττα Καλούδη,
για την πολύπλευρη στήριξη στη δύσκολη
περίοδο της συγγραφής

την Εύα Καρνάρη,
για την ψυχολογική στήριξη, την πρακτική
βοήθεια και την έμπνευση

και όλους τους φίλους και φίλες,
που ανέχτηκαν το άγχος, τα παράπονα
και τον ενθουσιασμό μου για την εργασία

για τους γονείς μου

Πίνακας Περιεχομένων

Πίνακας Εικόνων και Σχημάτων

ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Γενικά

Το αντικείμενο της διπλωματικής εργασίας αυτής είναι η **κατασκευή εφαρμογής βασισμένης στο πρωτόκολλο επικοινωνίας CAN** (Controller Area Network). Η εφαρμογή αποτελείται βασικά από δύο αυτόνομες «συσκευές», η επικοινωνία των οποίων πραγματοποιείται μέσω του συγκεκριμένου πρωτοκόλλου. Οι «συσκευές» αυτές (στο εξής θα αναφέρονται ως κόμβοι) έχουν ως πυρήνα από έναν 32-bit ARM μικροεπεξεργαστή με ενσωματωμένο σύστημα διαχείρισης του πρωτοκόλλου CAN. Για την επίδειξη της επικοινωνίας των δύο κόμβων, ο πρώτος συνδέεται μέσω σειριακής θύρας με προσωπικό υπολογιστή στον οποίο εκτελείται τερματικό πρόγραμμα ελέγχου των δεδομένων που μεταδίδονται αμφίδρομα στη σειριακή θύρα, ενώ ο δεύτερος είναι τοποθετημένος στο εσωτερικό της μινιατούρας ανελκυστήρα που κατασκευάστηκε και τοποθετήθηκε σε μακέτα δύο ορόφων και ελέγχει την κατάσταση των πλήκτρων ελέγχου, των αισθητήρων και των ενδείξεων του ανελκυστήρα καθώς και το βηματικό κινητήρα που τον μετακινεί, διαμέσου κατάλληλης ηλεκτρονικής διάταξης με την οποία συνδέεται μέσω καλωδίου.

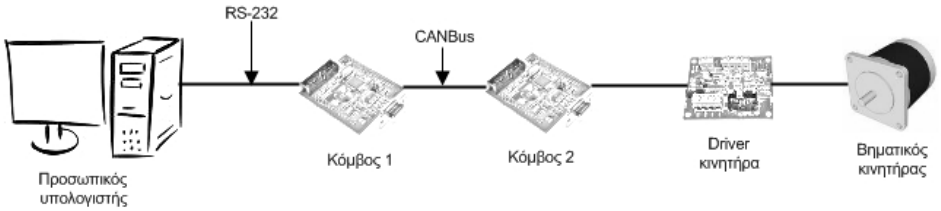
Η εφαρμογή αυτή αποτελεί προσομοίωση σε μικρή κλίμακα πραγματικών **εφαρμογών κατανεμημένου ελέγχου** οι οποίες αποτελούνται συνήθως από ένα πολυπληθές δίκτυο ελεγκτών. Οι ελεγκτές σε ένα τέτοιο σύστημα ελέγχου διαχειρίζονται κάποια επιμέρους διαδικασία, διεργασία ή ομάδα παραμέτρων ενώ ταυτόχρονα επικοινωνούν μεταξύ τους ώστε να γίνεται δυνατή τόσο η πραγμάτωση των συνολικότερων διεργασιών που κατανέμονται στις επιμέρους αυτές διεργασίες όσο και η συνολική εποπτεία και ο κεντρικός έλεγχος ολόκληρης της διαδικασίας. Στην εφαρμογή που κατασκευάζεται στα πλαίσια της διπλωματικής εργασίας αυτής, ο ένας από τους δύο κόμβους ελέγχει ψηφιακές εξόδους, ψηφιακές εισόδους και έναν ηλεκτρικό κινητήρα, δηλαδή μερικές από τις βασικότερες εργασίες που επιτελεί και ένας ελεγκτής σε μια πραγματική εφαρμογή ενώ ο δεύτερος κόμβος υλοποιεί τη σύνδεση του «δικτύου» των ελεγκτών (στην προκειμένη περίπτωση το δίκτυο αποτελείται από δύο μόνο ελεγκτές, το πρωτόκολλο CAN όμως υποστηρίζει πολύ μεγαλύτερα δίκτυα) με το κεντρικό σημείο εποπτείας και ελέγχου που είναι ένας προσωπικός υπολογιστής.

Στόχος της διπλωματικής εργασίας είναι η κατασκευή ενός ολοκληρωμένου συστήματος ελέγχου που προσομοιώνει πραγματικές εφαρμογές, συμπεριλαμβάνοντας

την περιγραφή της εφαρμογής, την επιλογή της αρχιτεκτονικής και του πρωτοκόλλου επικοινωνίας, το σχεδιασμό των κόμβων και των περιφερειακών που απαιτούνται, την κατασκευή αυτών και την ανάπτυξη του κατάλληλου κώδικα για τους μικροεπεξεργαστές που αποτελούν τον πυρήνα των κόμβων. Σκοπός είναι η εξοικείωση με το σχεδιασμό και την υλοποίηση συστημάτων ελέγχου που βασίζονται σε μικροεπεξεργαστή και η αντιμετώπιση των προβλημάτων που παρουσιάζονται σε μια τέτοια διαδικασία, ειδικά κατά τη μετάβαση από το θεωρητικό επίπεδο του σχεδιασμού στο πρακτικό επίπεδο της κατασκευής. Καίριες παράμετροι στην όλη διαδικασία είναι ο μέγιστος δυνατός **περιορισμός του κόστους** καθώς και η επιλογή πραγματοποιήσιμων μεθόδων και πρακτικών με δεδομένη την ένδοια πόρων και εργαλείων-κατάσταση η οποία επίσης προσομοιάζει τις απαιτήσεις που προβάλλονται σε πραγματικές καταστάσεις.

Αναλυτικά

Ένα συνοπτικό διάγραμμα της εφαρμογής φαίνεται στο παρακάτω σχήμα:



Εικόνα 1: Διάγραμμα της εφαρμογής

Εδώ θα περιγραφούν εν συντομία τα χαρακτηριστικά και η λειτουργία των διαφόρων μερών που απαρτίζουν την εφαρμογή ενώ στη συνέχεια του κειμένου θα δοθούν λεπτομέρειες.

Ο προσωπικός υπολογιστής μπορεί να είναι οποιαδήποτε συσκευή που μπορεί να προσφέρει αμφίδρομη επικοινωνία με το χρήστη και με δυνατότητα σειριακής επικοινωνίας μέσω του πρωτοκόλλου RS-232. Το πρωτόκολλο αυτό χρησιμοποιείται για τη διασύνδεση του χρήστη με τον Κεντρικό Κόμβο και επομένως με το δίκτυο CAN. Στο σημείο αυτό παρέχεται στο χρήστη πλήρης εποπτεία και έλεγχος του δικτύου, καθώς ο Κεντρικός Κόμβος αποτελεί το συνδετικό κρίκο μεταξύ του δικτύου CAN και του κεντρικού σημείου ελέγχου. Για την επίδειξη του συστήματος χρησιμοποιείται φορητός

υπολογιστής με λειτουργικό σύστημα Windows XP και λογισμικό ελέγχου της σειριακής θύρας που έχει ενσωματωμένη ο υπολογιστής. Το λογισμικό αυτό παρέχει ταυτόχρονα λειτουργία τερματικού και τη δυνατότητα προγραμματισμού των μικροεπεξεργαστών των κόμβων.

Ο Κεντρικός Κόμβος αποτελείται από μία διάτρητη πλακέτα όπου έχει τοποθετηθεί τυπωμένη πλακέτα βασισμένη στον μικροεπεξεργαστή LPC2129 της Philips Semiconductors και ακροδέκτες σύνδεσης με την πηγή τάσης και με το δίκτυο CAN. Επιπλέον, στη διάτρητη πλακέτα έχουν τοποθετηθεί ένα πλήκτρο επανεκκίνησης του μικροεπεξεργαστή και τρία πλήκτρα που αντιστοιχούν σε κλήση του ανελκυστήρα από τους τρεις ορόφους της μακέτας, συνοδευόμενα από τις απαραίτητες αντιστάσεις. Η τυπωμένη πλακέτα (header board), που κατασκευάζεται από την εταιρία Embedded Artists, περιλαμβάνει έναν LPC2129, μια θύρα RS-232 τύπου DSUB-9 και μια θύρα JTAG καθώς και όλα τα απαραίτητα περιφερειακά ολοκληρωμένα διευκολύνοντας σημαντικά την ανάπτυξη της εφαρμογής του μικροεπεξεργαστή για το χρήστη. Συγκεκριμένα, δίνεται η δυνατότητα προγραμματισμού του LPC2129 μέσω της σειριακής θύρας με τη χρήση κατάλληλου λογισμικού, προσφέρεται φυσική διασύνδεση με όλους τους ακροδέκτες του μικροεπεξεργαστή σε ακροδέκτες τυποποιημένου μεγέθους που μπορούν να χρησιμοποιηθούν σε διάτρητη πλακέτα και επιτρέπεται η σύνδεση του ολοκληρωμένου σε μία πηγή τάσης 5V αφού περιλαμβάνονται ολοκληρωμένα που μετατρέπουν την τάση αυτή σε 3,3V και 1,8V, τα επίπεδα τάσης που απαιτούνται για τη λειτουργία του μικροεπεξεργαστή. Λειτουργικά, ο Κεντρικός Κόμβος διεκπεραιώνει τη σύνδεση του δικτύου CAN με τη σειριακή θύρα και άρα με το χρήστη, από τον οποίο λαμβάνει μηνύματα τα οποία διαβιβάζει στο δίκτυο CAN και στον οποίο μεταβιβάζει τα μηνύματα που λαμβάνονται από το δίκτυο CAN.

Ο Κόμβος Ανελκυστήρα αποτελείται από μία πανομοιότυπη τυπωμένη πλακέτα (header board) με αυτήν του Κεντρικού Κόμβου, η οποία έχει τοποθετηθεί σε διάτρητη πλακέτα που επίσης περιλαμβάνει τους αντιστροφείς και τις αντιστάσεις που απαιτούνται για τη σύνδεση των πλήκτρων, του αισθητήρα και των ενδεικτικών LED του ανελκυστήρα. Στη διάτρητη πλακέτα του Κόμβου Ανελκυστήρα περιλαμβάνονται, τέλος, οι αναγκαίοι ακροδέκτες για τη σύνδεση με την πηγή τάσης, το δίκτυο CAN, τα συστήματα του ανελκυστήρα και τον driver του ηλεκτρικού βηματικού κινητήρα. Η λειτουργία του Κόμβου Ανελκυστήρα είναι ο έλεγχος των ψηφιακών εισόδων και εξόδων του ανελκυστήρα και ταυτόχρονα ο έλεγχος του βηματικού κινητήρα διαμέσου του driver

με τη χρήση ενός ψηφιακού σήματος κατεύθυνσης και ενός ψηφιακού σήματος παλμών που προκαλεί την περιστροφή του κινητήρα. Ο έλεγχος αυτός γίνεται ανάλογα με τις εντολές που δέχεται ο κόμβος είτε τοπικά από τα πλήκτρα που βρίσκονται στον ανελκυστήρα είτε από το δίκτυο CAN το οποίο με τη σειρά του ελέγχεται διαμέσου του Κεντρικού Κόμβου από το χρήστη.

Ο ανελκυστήρας είναι κατασκευασμένος από χαρτόνι και τοποθετημένος σε μακέτα δύο ορόφων, κατασκευασμένη από χαρτόνι και ξύλο. Κινείται επομένως μεταξύ τριών θέσεων – ισογείου, πρώτου και δεύτερου ορόφου. Πάνω του βρίσκονται τρία πλήκτρα που αντιστοιχούν στις τρεις δυνατές θέσεις του, ένα πλήκτρο διακοπής κίνησης (STOP), ένας διακόπτης κινδύνου δύο θέσεων (ALARM) και ένας διακόπτης δύο θέσεων που προσομοιώνει τον αισθητήρα βάρους που βρίσκεται σε πραγματικούς ανελκυστήρες και δηλώνει αν ο ανελκυστήρας είναι γεμάτος ή όχι. Ως ενδείξεις, υπάρχει ένα LED κινδύνου που διατηρείται αναμμένο όσο ο διακόπτης κινδύνου είναι ενεργοποιημένος και ένα 7-segment display όπου εμφανίζεται ως αριθμός η τρέχουσα θέση. Τέλος, στο κάτω μέρος του εσωτερικού του ανελκυστήρα βρίσκεται ένας μαγνητικός αισθητήρας συνδεδεμένος με ένα ενδεικτικό LED, ο οποίος εμφανίζει μηδενική αντίσταση όταν πλησιάσει σε έναν από τους τρεις μόνιμους μαγνήτες που έχουν τοποθετηθεί στους ορόφους και άπειρη διαφορετικά. Η μακέτα περιλαμβάνει το ξύλινο σύστημα στήριξης (κολώνες), τις χάρτινες πλάκες των ορόφων και τρία χάρτινα τοιχία που δομούν το φρεάτιο του ανελκυστήρα. Στο επάνω μέρος του φρεατίου είναι τοποθετημένος ο βηματικός κινητήρας, στον άξονα του οποίου συνδέεται με κλέμα ένας μεγαλύτερος άξονας, όπου τυλίγεται το σκοινί με το οποίο μετακινείται ο ανελκυστήρας. Στα πλαϊνά τοιχία του φρεατίου έχουν τοποθετηθεί ράγες από χαρτόνι κατά μήκος των οποίων κινείται ο ανελκυστήρας έχοντας στα αντίστοιχα σημεία του εξοχές, ώστε η θέση του να είναι πάντα κατακόρυφη και η κίνησή του ευθύγραμμη δίχως παρεκκλίσεις.

Ο driver του κινητήρα είναι μια διάτρητη πλακέτα που περιλαμβάνει κάποια ολοκληρωμένα, τους αναγκαίους ακροδέκτες για τη σύνδεση με τα σήματα ελέγχου ως είσοδο και το βηματικό κινητήρα ως έξοδο και ένα σύστημα διακοπτών για τη χειροκίνητη λειτουργία του. Ο driver φροντίζει να «οδηγεί» το κατάλληλο ρεύμα που απαιτείται για την κίνηση του κινητήρα και ταυτόχρονα περιλαμβάνει τη λογική που του επιτρέπει τον έλεγχο του κινητήρα από δύο ψηφιακά σήματα: ένα σήμα επιπέδου τάσης που ορίζει την κατεύθυνση περιστροφής του κινητήρα και ένα σήμα παλμών που προκαλεί την περιστροφή του κατά βήματα. Τα δύο αυτά σήματα παράγονται είτε

χειροκίνητα από δύο διακόπτες (ένα διακόπτη δύο θέσεων και ένα push-button) είτε λαμβάνονται από εξωτερική πηγή, δηλαδή στη συγκεκριμένη εφαρμογή από τον Κόμβο Ανελκυστήρα.

Ο κινητήρας είναι ένας διπολικός διφασικός βηματικός ηλεκτρικός DC κινητήρας. Αποτελείται από δύο πηνία τοποθετημένα με τέτοιο τρόπο ώστε η διαρροή τους από ρεύμα κατάλληλης έντασης και η αλλαγή της φοράς αυτού, ακολουθώντας μια συγκεκριμένη ακολουθία, να προκαλεί την περιστροφή του άξονα κατά ένα συγκεκριμένο αριθμό μοιρών. Μία τέτοια περιστροφή ονομάζεται βήμα· εξ ου και η ονομασία του κινητήρα. Τέτοιοι κινητήρες υπάρχουν σε διάφορα μεγέθη παράγοντας αντίστοιχη ροπή και απαιτώντας αντίστοιχη ένταση ρεύματος. Χρησιμοποιούνται σε εφαρμογές όπου απαιτείται η τοποθέτηση του άξονα σε πεπερασμένο αριθμό συγκεκριμένων θέσεων, προσφέροντας έλεγχο θέσης χωρίς την ανάγκη ανάδρασης (feedback). Η συγκεκριμένη εφαρμογή μολαταύτα απαιτεί ένα διακριτό σήμα ανάδρασης ώστε ο ελεγκτής να γνωρίζει πότε ο ανελκυστήρας βρίσκεται σε όροφο και πότε μεταξύ ορόφων. Το σήμα αυτό παράγεται από μαγνητική επαφή, ελέγχωντας διαρκώς την κατάσταση του μαγνητικού αισθητήρα που είναι τοποθετημένος στο εσωτερικό του ανελκυστήρα και ο οποίος συμπεριφέρεται ως ανοιχτοκύκλωμα όταν βρίσκεται μεταξύ ορόφων και ως βραχυκύκλωμα όταν βρίσκεται σε μαγνητική σύζευξη με έναν από τους μόνιμους μαγνήτες που έχουν τοποθετηθεί σε κάθε όροφο.

ΚΟΜΒΟΙ CAN

Γενικά

Η μαζική παραγωγή μίας ηλεκτρονικής συσκευής που βασίζεται σε μικροεπεξεργαστή, σε πραγματικά δεδομένα, προϋποθέτει ένα στάδιο ανάπτυξης και μελέτης της εφαρμογής. Στο στάδιο αυτό, τυπικά, κατασκευάζεται μία πρότυπη συσκευή (prototype) μέσω της οποίας μελετώνται τα χαρακτηριστικά κυρίως του επεξεργαστή που αποτελεί τη βάση της εφαρμογής και γίνεται ο σχεδιασμός και η ανάπτυξη του λογισμικού που απαιτείται να εκτελεί ο επεξεργαστής ώστε η πρότυπη συσκευή να πραγματοποιεί τις επιθυμητές λειτουργίες. Ταυτόχρονα, μελετώνται και σχεδιάζονται ή επιλέγονται τα περιφερειακά εκείνα που είναι αναγκαία για τη σωστή λειτουργία της συσκευής στα πλαίσια των προδιαγραφών της. Οι προδιαγραφές αυτές περιλαμβάνουν, τουλάχιστον, την τάση τροφοδοσίας και τον τρόπο διασύνδεσης της συσκευής με το περιβάλλον, δηλαδή τα χαρακτηριστικά των εισόδων και εξόδων του συστήματος. Η ενδεικτική εφαρμογή που κατασκευάζεται στα πλαίσια της εργασίας αυτής απαιτεί την τυποποιημένη τροφοδοσία τάσης 5V για τα λογικά κυκλώματα ενώ οι είσοδοι και έξοδοι αναλύονται στη συνέχεια για κάθε τμήμα της εφαρμογής.

Οι δύο κόμβοι του δικτύου CAN που υλοποιείται βασίζονται στο μικροεπεξεργαστή LPC2129 της Philips Semiconductors. Η συγκεκριμένη επιλογή έγινε με βάση τα κύρια χαρακτηριστικά του επεξεργαστή, δηλαδή ο 32-bit ARM7TDMI-S πυρήνας με τα 16kB στατικής RAM που του προσφέρει σημαντική επεξεργαστική ισχύ ενώ παράλληλα διευκολύνει την ανάπτυξη κώδικα καθώς υπάρχουν πολλά διαθέσιμα εργαλεία για πυρήνες ARM, η ύπαρξη 256kB ενσωματωμένης μνήμης flash για την αποθήκευση του κώδικα, η δυνατότητα προγραμματισμού του χωρίς την αφαίρεσή του από το σύστημα, οι ενσωματωμένοι ελεγκτές CAN και το πλήθος των διαθέσιμων εισόδων και εξόδων, συμπεριλαμβανομένων διασυνδέσεων τύπου UART και ψηφιακών εισόδων-εξόδων γενικής χρήσης. Επιπλέον, ο μικροεπεξεργαστής διατίθεται σε συσκευασία LQFP64 με μικρό κόστος (μικρότερο από 10$ το κομμάτι) και, το κυριότερο, υπάρχουν πολλά εργαλεία ανάπτυξης γύρω από αυτόν.

Προτού αναλυθούν οι επιμέρους λεπτομέρειες των δύο κόμβων, θα γίνει μια παρουσίαση του σημαντικότερου κοινού τους κυκλώματος.

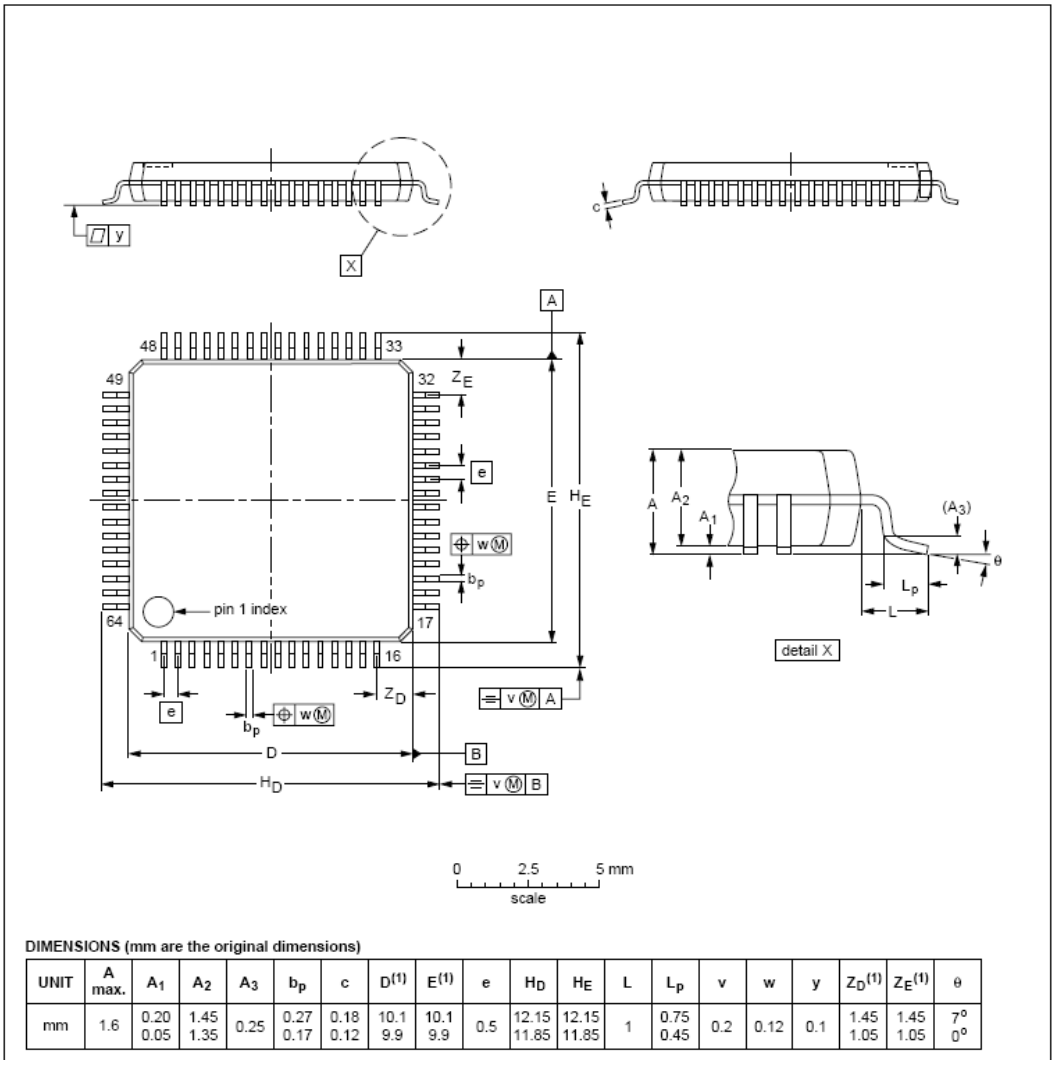
ΤΥΠΩΜΕΝΗ ΠΛΑΚΕΤΑ (HEADER BOARD)

Γενικά

Τα υλικά εργαλεία που χρησιμοποιούνται κατά την ανάπτυξη της πρότυπης συσκευής για τη μελέτη του μικροεπεξεργαστή και τη διαμόρφωση του αντίστοιχου λογισμικού του διατίθενται σε διάφορα επίπεδα ολοκλήρωσης με το αντίστοιχο κόστος αλλά και διευκόλυνση.

Το κατώτερο επίπεδο είναι αυτούσιο το chip, δηλαδή ο μικροεπεξεργαστής στη συσκευασία LQFP64 που διατίθεται από τη Philips. Αποτελεί δε την πιο οικονομική λύση.

LQFP64: plastic low profile quad flat package; 64 leads; body 10 x 10 x 1.4 mm **SOT314-2**



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_D	H_E	L	L_p	v	w	y	$Z_D^{(1)}$	$Z_E^{(1)}$	θ
mm	1.6	0.20 0.05	1.45 1.35	0.25	0.27 0.17	0.18 0.12	10.1 9.9	10.1 9.9	0.5	12.15 11.85	12.15 11.85	1	0.75 0.45	0.2	0.12	0.1	1.45 1.05	1.45 1.05	7° 0°

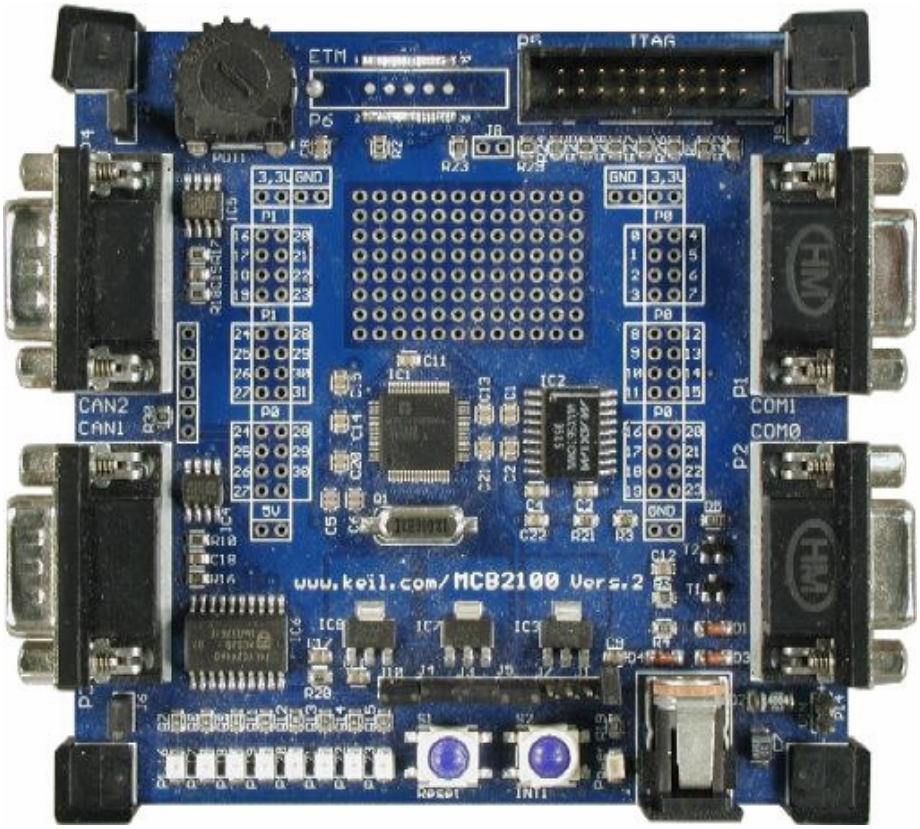
Εικόνα 2: Συσκευασία LQFP64 [1]

Η συσκευασία αυτή είναι κατάλληλη μόνο για SMD (Surface Mount Devices) κατασκευές, οι οποίες απαιτούν ιδιαίτερες μεθόδους κατασκευής με χρήση κατάλληλων φούρνων και άλλων συσκευών που χρησιμοποιούνται στη βιομηχανική μαζική παραγωγή πλακετών και επομένως είναι αδύνατη η χρήση της αυτούσιας για τους σκοπούς της παρούσας εργασίας.

Το επόμενο επίπεδο που διατίθεται στην αγορά αποτελείται από μια τυπωμένη πλακέτα μικρού μεγέθους που περιλαμβάνει το μικροεπεξεργαστή στην παραπάνω συσκευασία του, κάποιες θύρες επικοινωνίας και τα απαραίτητα περιφερειακά ώστε όλοι οι ακροδέκτες του μικροεπεξεργαστή να είναι συνδεδεμένοι με τους ακροδέκτες της τυπωμένης πλακέτας. Οι τελευταίοι αυτοί ακροδέκτες είναι τυποποιημένου μεγέθους και απόστασης, τέτοιων ώστε να είναι δυνατή η τοποθέτηση της ολοκληρωμένης πλακέτας σε διάτρητη πλακέτα. Αυτό το επίπεδο ολοκλήρωσης ουσιαστικά παρέχει πλήρη πρόσβαση στους ακροδέκτες του υπό εξέταση επεξεργαστή και ταυτόχρονα περιλαμβάνει τα αναγκαία περιφερειακά ώστε η τροφοδοσία να παρέχεται από πηγή τάσης 5V και η χρήση κάποιων ενσωματωμένων στον επεξεργαστή περιφερειακών να γίνεται απ' ευθείας (συνήθως πρόκειται για μία θύρα RS-232 και μία θύρα JTAG). Οι παραπάνω ιδιότητες όπως και η ονομασία της τυπωμένης πλακέτας μπορεί να μεταβάλλονται σε λεπτομέρειες από κατασκευαστή σε κατασκευαστή αλλά με βάση τη συνηθισμένη ορολογία λέγονται «Header Boards». Οι τυπωμένες πλακέτες τύπου «Header Board» συνήθως τοποθετούνται σε διάτρητες πλακέτες όπου βρίσκονται τα υπόλοιπα ολοκληρωμένα που απαιτούνται για την κατασκευή της πρότυπης συσκευής. Μία τέτοια λύση χρησιμοποιείται στους δύο κόμβους CAN της εφαρμογής. Έχει κατασκευαστεί από την εταιρία Embedded Artists AB και διατίθεται με το όνομα «LPC2129 CAN QuickStart Board» στην τιμή των 60€, που είναι αρκετά προσιτή.

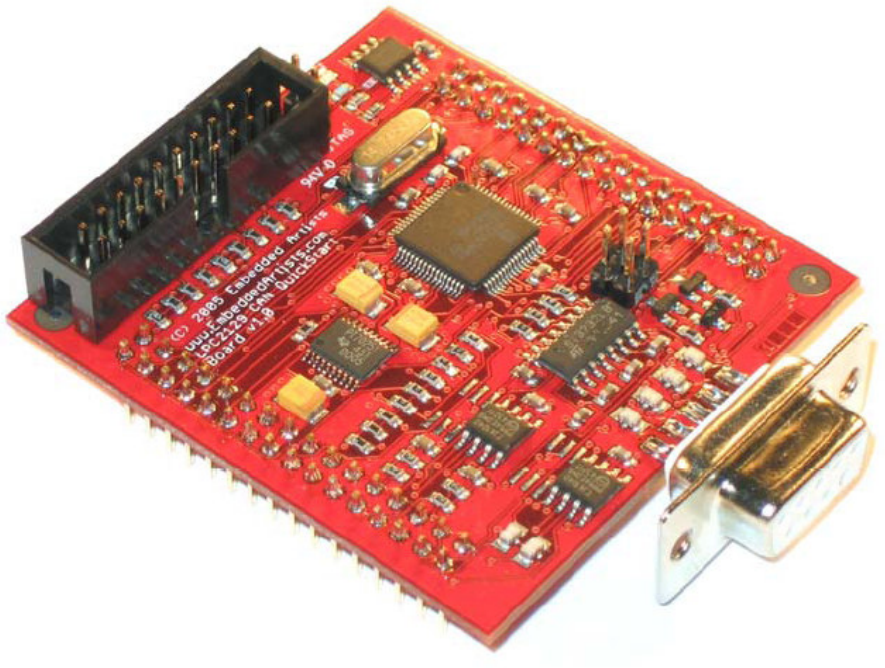
Το ανώτερο στάδιο ολοκλήρωσης σε εργαλεία ανάπτυξης αποτελούν οι κοινώς αποκαλούμενες «Αναπτυξιακές Πλακέτες» («Development Boards»). Αυτές είναι τυπωμένες πλακέτες μεγαλύτερου μεγέθους από τις προηγούμενες που έχουν ως πυρήνα τον υπό εξέταση μικροεπεξεργαστή αλλά περιλαμβάνουν και μια σειρά από δυνατότητες επικοινωνίας με το χρήστη σε φυσικό πλέον επίπεδο. Αν και οι δυνατότητες αυτές μεταβάλλονται σημαντικά ανάμεσα στους διάφορους κατασκευαστές, τυπικά σε μια «Αναπτυξιακή Πλακέτα» θα συναντήσει κανείς από LEDs και διακόπτες, θύρες επικοινωνίας (JTAG, RS-232, CANBus ή άλλες θύρες ανάλογα με την ιδιαίτερη χρήση

για την οποία προορίζεται ο επεξεργαστής), τυποποιημένους ακροδέκτες για σύνδεση με τροφοδοτικά, μέχρι και μικρές LCD οθόνες και πληκτρολόγια. Το κόστος μιας τέτοιας πλακέτας μεταβάλλεται εξαιρετικά ανάλογα με τα χαρακτηριστικά της αλλά σε κάθε περίπτωση οι τιμές ξεκινούν από περίπου 400€. Συχνά, οι τυπωμένες «Αναπτυξιακές Πλακέτες» συνοδεύονται από πακέτα ανάπτυξης λογισμικού για τους αντίστοιχους μικροεπεξεργαστές, ανεβάζοντας με τον τρόπο αυτό το κόστος μέχρι ακόμα και μερικές χιλιάδες ευρώ. Αν και οι διευκολύνσεις που παρέχονται από τέτοιες λύσεις είναι εξαιρετικές, το κόστος κάνει τη χρήση τους αδύνατη στα πλαίσια της διπλωματικής αυτής εργασίας, με δεδομένο ειδικά ότι απαιτούνται δύο κόμβοι για το δίκτυο CAN.



Εικόνα 3: Αναπτυξιακή πλακέτα MCB2100 της Keil [2]

Όπως αναφέρθηκε και παραπάνω, για την υλοποίηση των κόμβων CAN της εφαρμογής χρησιμοποιείται μία «header board» της εταιρείας Embedded Artists AB. Αυτή φαίνεται στην παρακάτω εικόνα.

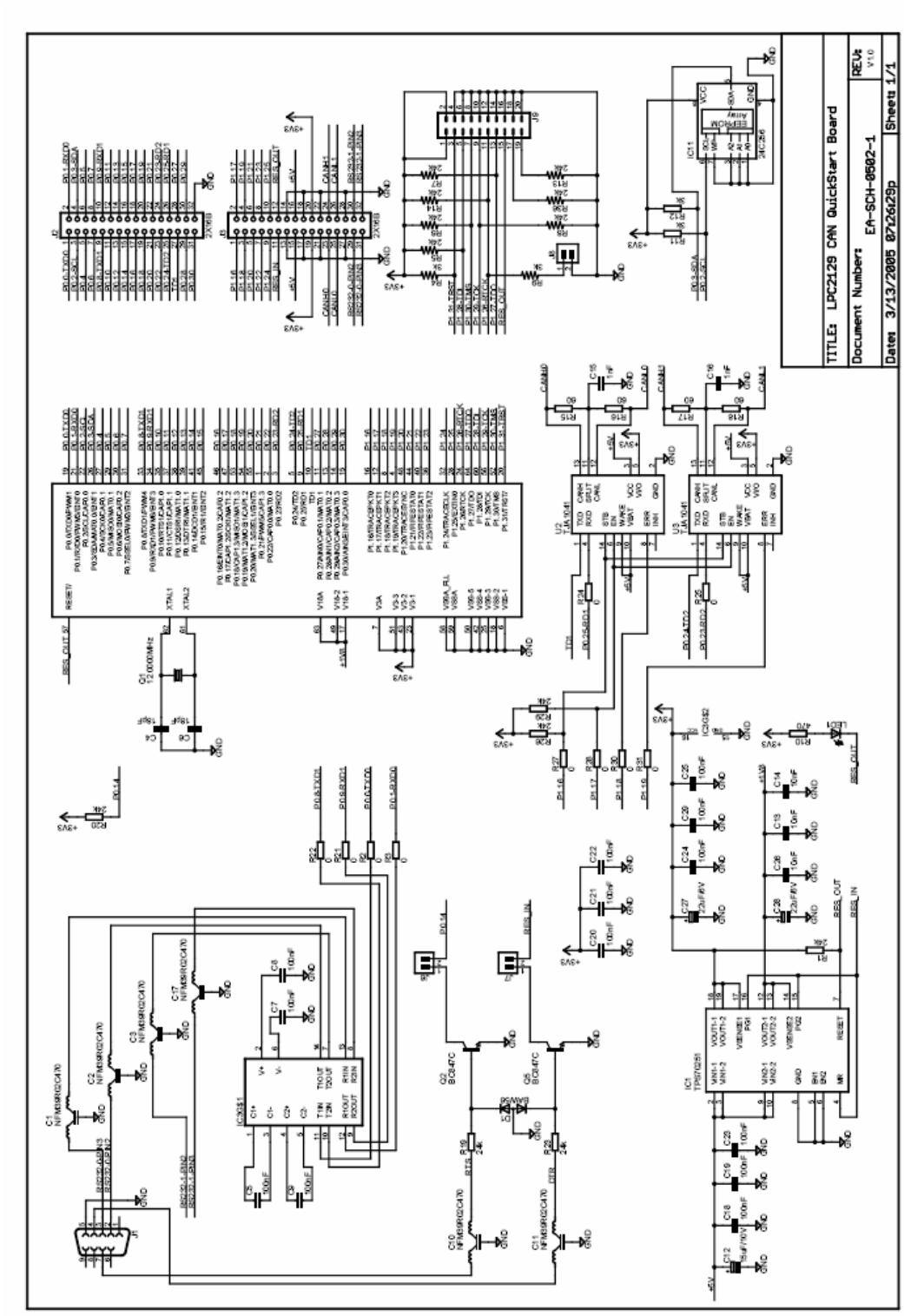


Εικόνα 4: LPC2129 CAN QuickStart Board [3]

Αμέσως διακρίνονται η θύρα RS-232 σε θηλυκό DSUB9 δεξιά και η θύρα JTAG αριστερά, οι οποίες προσφέρουν δυνατότητες επικοινωνίας και ελέγχου του επεξεργαστή μέσω των αντίστοιχων πρωτοκόλλων. Στο κέντρο περίπου είναι τοποθετημένος ο μικροεπεξεργαστής LPC2129 που αποτελεί τον πυρήνα της πλακέτας. Δίπλα του διακρίνεται ο κρυσταλλος που σε συνεργασία με τον κρυσταλλικό ταλαντωτή του επεξεργαστή παράγει τους παλμούς ρολογιού για τον πυρήνα και τα ενσωματωμένα περιφερειακά. Στα αριστερά του επεξεργαστή υπάρχει το ολοκληρωμένο TPS70251 που αναλαμβάνει την παραγωγή των απαραίτητων επιπέδων τάσης από την τάση τροφοδοσίας που είναι στα 5V και επίσης παράγει κατάλληλα το σήμα επανεκκίνησης που είναι συνδεδεμένο με τον αντίστοιχο ακροδέκτη του LPC2129 δίνοντας ταυτόχρονα το σήμα αυτό ως έξοδο σε ακροδέκτη της τυπωμένης πλακέτας ώστε να μπορεί να χρησιμοποιηθεί για το συγχρονισμό άλλων συσκευών. Μπροστά από τη θύρα RS-232 βρίσκεται το ολοκληρωμένο IC3G$1 με λειτουργία αντίστοιχη του γνωστού MAX232 που φροντίζει για τη μετάβαση του σήματος από τη στάθμη που βρίσκονται οι ακροδέκτες του επεξεργαστή στη στάθμη που ορίζεται για τη σύνδεση με τη γραμμή μεταφοράς σύμφωνα με τις απαιτήσεις του σειριακού πρωτοκόλλου RS-232. Κάτω αριστερά διακρίνονται δύο όμοια ολοκληρωμένα, τα TJA1041, τα οποία με τη σειρά τους επιτρέπουν τη διασύνδεση των ακροδεκτών του επεξεργαστή που σχετίζονται με το

σύστημα CAN με το δίαυλο του δικτύου με βάση τις απαιτήσεις για τη φυσική διασύνδεση. Τέλος, στην επάνω δεξιά γωνία υπάρχει το μικρό ολοκληρωμένο 24C256 που είναι μνήμη τύπου E^2PROM και χρησιμοποιείται στο σύστημα σειριακής επικοινωνίας I^2C, το οποίο δεν θα απασχολήσει τη συγκεκριμένη εφαρμογή.

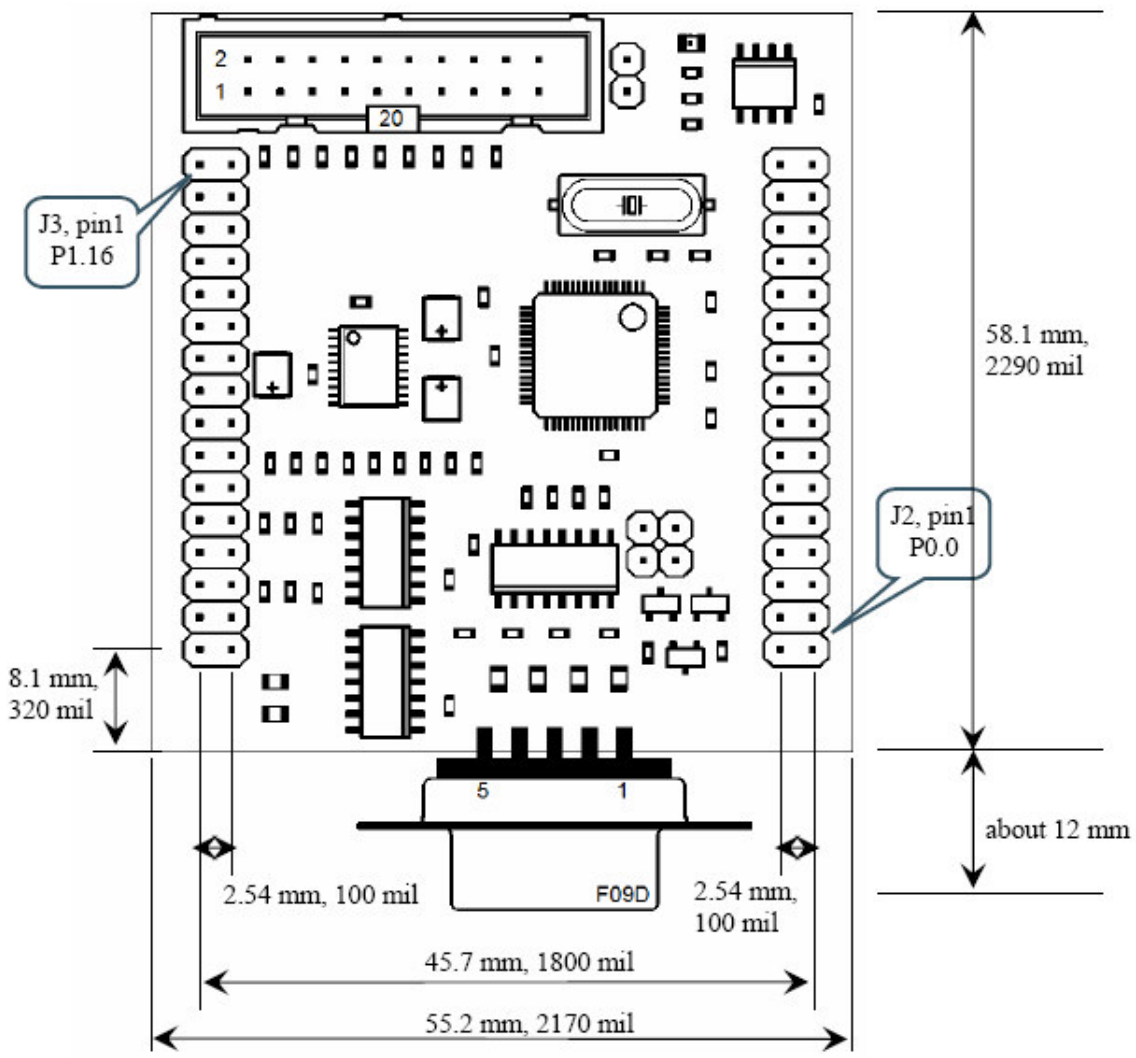
Το σχηματικό της τυπωμένης πλακέτας δίνεται στην παρακάτω εικόνα.



Εικόνα 5: Σχηματικό της τυπωμένης πλακέτας «Header Board» [3]

Στο σχηματικό αυτό περιλαμβάνονται και οι ακροδέκτες τυποποιημένου μεγέθους και απόστασης (2,54 mm) για την τοποθέτηση της πλακέτας σε διάτρητη, ομαδοποιημένοι σε δύο ομάδες των 2x16 ακροδεκτών.

Το μηχανολογικό σχέδιο της τυπωμένης πλακέτας είναι το εξής:



Εικόνα 6: Μηχανολογικό σχέδιο της τυπωμένης πλακέτας «Header Board» [3]

Στη συνέχεια του κειμένου θα αναλυθούν τα χαρακτηριστικά των ολοκληρωμένων που αναφέρθηκαν, με ιδιαίτερη έμφαση στο μικροεπεξεργαστή.

ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΗΣ PHILIPS LPC2129

Γενικά

Ο LPC2129 είναι μέλος της οικογένειας μικροεπεξεργαστών LPC2000 που περιλαμβάνει τους LPC2119/2129/2194/2292/2294 της Philips Semiconductors. Η οικογένεια αυτή μοιράζεται κάποια χαρακτηριστικά. Πρώτα και κύρια, η Κεντρική Μονάδα Επεξεργασίας (CPU) είναι τύπου ARM7TDMI-S, δηλαδή η πλέον εξελιγμένη έκδοση του πυρήνα ARM7, ο οποίος αποτελεί έναν RISC (Reduced Instruction Set Computer), γεγονός που σημαίνει ότι η απλότητα του σχεδιασμού του επιτρέπει την υλοποίησή του με συγκριτικά μικρό αριθμό πυλών. Αυτό με τη σειρά του συνεπάγεται υψηλή απόδοση, μικρή κατανάλωση ισχύος και μικρές απαιτήσεις σε χώρο, επιτρέποντας τη συνύπαρξη πληθώρας περιφερειακών καθώς και μνήμης στο ίδιο chip πυριτίου. Ταυτόχρονα, υπάρχει διαθέσιμο πλήθος εργαλείων για τον προγραμματισμό του πυρήνα αυτού επιτρέποντας την ανάπτυξη κώδικα σε γλώσσα υψηλού επιπέδου αφήνοντας τις λεπτομέρειες του συνόλου των εντολών του σε κάποιον από τους πολλούς compilers και assemblers που κυκλοφορούν από μηδενικό έως αρκετά σημαντικό κόστος.

Εξαιτίας του ARM πυρήνα του, ο LPC2129 μπορεί να λειτουργήσει τόσο ως 32-bit όσο και ως 16-bit επεξεργαστής, με την πρώτη επιλογή να προσφέρει καλύτερες επιδόσεις και τη δεύτερη μικρότερη απαιτούμενη ποσότητα κώδικα. Επιπλέον, ο μικροεπεξεργαστής αυτός διαθέτει 16kB στατικής μνήμης RAM και 256kB μνήμης flash διαθέσιμης για την αποθήκευση του προγράμματος, αυξημένες δυνατότητες ελέγχου της εκτέλεσης των εντολών κατά τη διάρκεια της λειτουργίας του και πολλά περιφερειακά συμπεριλαμβανομένων:

- 2 διασυνδεδεμένων ελεγκτών CAN
- 2 ελεγκτών UART για σειριακή επικοινωνία
- 10-bit μετατροπέα Α/Ψ τεσσάρων καναλιών
- 2 32-bit μετρητών
- μονάδας PWM
- ελεγκτών σειριακής επικοινωνίας τύπου SPI και I^2C
- 46 ακροδεκτών γενικής χρήσης εισόδου/εξόδου

Τέλος, ο LPC2129 υποστηρίζει καταστάσεις μειωμένης κατανάλωσης ισχύος τόσο για τον πυρήνα όσο και για καθένα από τα περιφερειακά, προηγμένο σύστημα διακοπών, προγραμματισμό της ενσωματωμένης μνήμης flash και διαμέσου του ελεγκτή UART

χωρίς να απαιτείται η αφαίρεσή του από το σύστημα και μέγιστη συχνότητα λειτουργίας τα 60MHz.

Το εύρος θερμοκρασίας λειτουργίας βρίσκεται μεταξύ των -40 και +85 βαθμών C ενώ οι ελάχιστες και μέγιστες τιμές των χαρακτηριστικών λειτουργίας συνοψίζονται στον παρακάτω πίνακα:

Symbol	Parameter	Conditions		Min	Max	Unit
$V_{DD(1V8)}$	supply voltage (1.8 V)		[2]	–0.5	+2.5	V
$V_{DD(3V3)}$	supply voltage (3.3 V)		[3]	–0.5	+3.6	V
$V_{DDA(3V3)}$	analog supply voltage (3.3 V)			–0.5	+4.6	V
V_{IA}	analog input voltage			–0.5	+5.1	V
V_I	input voltage	5 V tolerant I/O pins	[4][5]	–0.5	+6.0	V
		other I/O pins	[4][6]	–0.5	$V_{DD(3V3)}$ + 0.5	V
I_{DD}	supply current		[7][8]	-	100	mA
I_{SS}	ground current		[8][9]	-	100	mA
T_{stg}	storage temperature		[10]	–65	+150	°C
$P_{tot(pack)}$	total power dissipation (per package)	based on package heat transfer, not device power consumption		-	1.5	W
V_{esd}	electrostatic discharge voltage	human body model	[11]			
		all pins		–2000	+2000	V
		machine model	[12]			
		all pins		–200	+200	V

Πίνακας 1: Ελάχιστες και μέγιστες επιτρεπτές τιμές λειτουργίας [4]

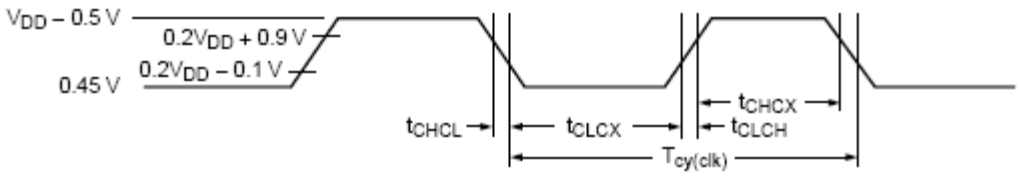
Τα χαρακτηριστικά στατικής και δυναμικής λειτουργίας δίνονται στους αντίστοιχους πίνακες παρακάτω:

Symbol	Parameter	Conditions		Min	Typ[1]	Max	Unit
$V_{DD(1V8)}$	supply voltage (1.8 V)		[2]	1.65	1.8	1.95	V
$V_{DD(3V3)}$	supply voltage (3.3 V)		[3]	3.0	3.3	3.6	V
$V_{DDA(3V3)}$	analog supply voltage (3.3 V)			2.5	3.3	3.6	V
Standard port pins, $\overline{\text{RESET}}$, RTCK							
I_{IL}	LOW-state input current	V_I = 0 V; no pull-up		-	-	3	µA
I_{IH}	HIGH-state input current	$V_I = V_{DD(3V3)}$; no pull-down		-	-	3	µA
I_{OZ}	OFF-state output current	V_O = 0 V; $V_O = V_{DD(3V3)}$; no pull-up/down		-	-	3	µA
I_{latch}	I/O latch-up current	$-(0.5V_{DD(3V3)}) < V_I < (1.5V_{DD(3V3)})$; T_j < 125 °C		100	-	-	mA
V_I	input voltage		[4][5][6]	0	-	5.5	V
V_O	output voltage	output active		0	-	$V_{DD(3V3)}$	V
V_{IH}	HIGH-state input voltage			2.0	-	-	V
V_{IL}	LOW-state input voltage			-	-	0.8	V
V_{hys}	hysteresis voltage			-	0.4	-	V
V_{OH}	HIGH-state output voltage	I_{OH} = –4 mA	[7]	$V_{DD(3V3)} - 0.4$	-	-	V
V_{OL}	LOW-state output voltage	I_{OL} = –4 mA	[7]	-	-	0.4	V
I_{OH}	HIGH-state output current	$V_{OH} = V_{DD(3V3)} - 0.4$ V	[7]	–4	-	-	mA
I_{OL}	LOW-state output current	V_{OL} = 0.4 V	[7]	4	-	-	mA
I_{OHS}	HIGH-state short-circuit output current	V_{OH} = 0 V	[9]	-	-	–45	mA
I_{OLS}	LOW-state short-circuit output current	$V_{OL} = V_{DD(3V3)}$	[9]	-	-	50	mA
I_{pd}	pull-down current	V_I = 5 V	[10]	10	50	150	µA
I_{pu}	pull-up current	V_I = 0 V	[11]	–15	–50	–85	µA
		$V_{DD(3V3)} < V_I$ < 5 V	[10]	0	0	0	µA
$I_{DD(act)}$	active mode supply current	$V_{DD(1V8)}$ = 1.8 V; CCLK = 60 MHz; T_{amb} = 25 °C; code `while(1){}` executed from flash; no active peripherals		-	60	-	mA
$I_{DD(pd)}$	Power-down mode supply current	$V_{DD(1V8)}$ = 1.8 V; T_{amb} = 25 °C		-	10	-	µA
		$V_{DD(1V8)}$ = 1.8 V; T_{amb} = 85 °C		-	110	500	µA
I^2C-bus pins							
V_{IH}	HIGH-state input voltage			$0.7V_{DD(3V3)}$	-	-	V
V_{IL}	LOW-state input voltage			-	-	$0.3V_{DD(3V3)}$	V
V_{hys}	hysteresis voltage			-	$0.5V_{DD(3V3)}$	-	V
V_{OL}	LOW-state output voltage	I_{OLS} = 3 mA	[7]	-	-	0.4	V
I_{LI}	input leakage current	$V_I = V_{DD(3V3)}$	[8]	-	2	4	µA
		V_I = 5 V		-	10	22	µA
Oscillator pins							
$V_{i(XTAL1)}$	input voltage on pin XTAL1			0	-	1.8	V
$V_{o(XTAL2)}$	output voltage on pin XTAL2			0	-	1.8	V

Πίνακας 2: Χαρακτηριστικά στατικής λειτουργίας [4]

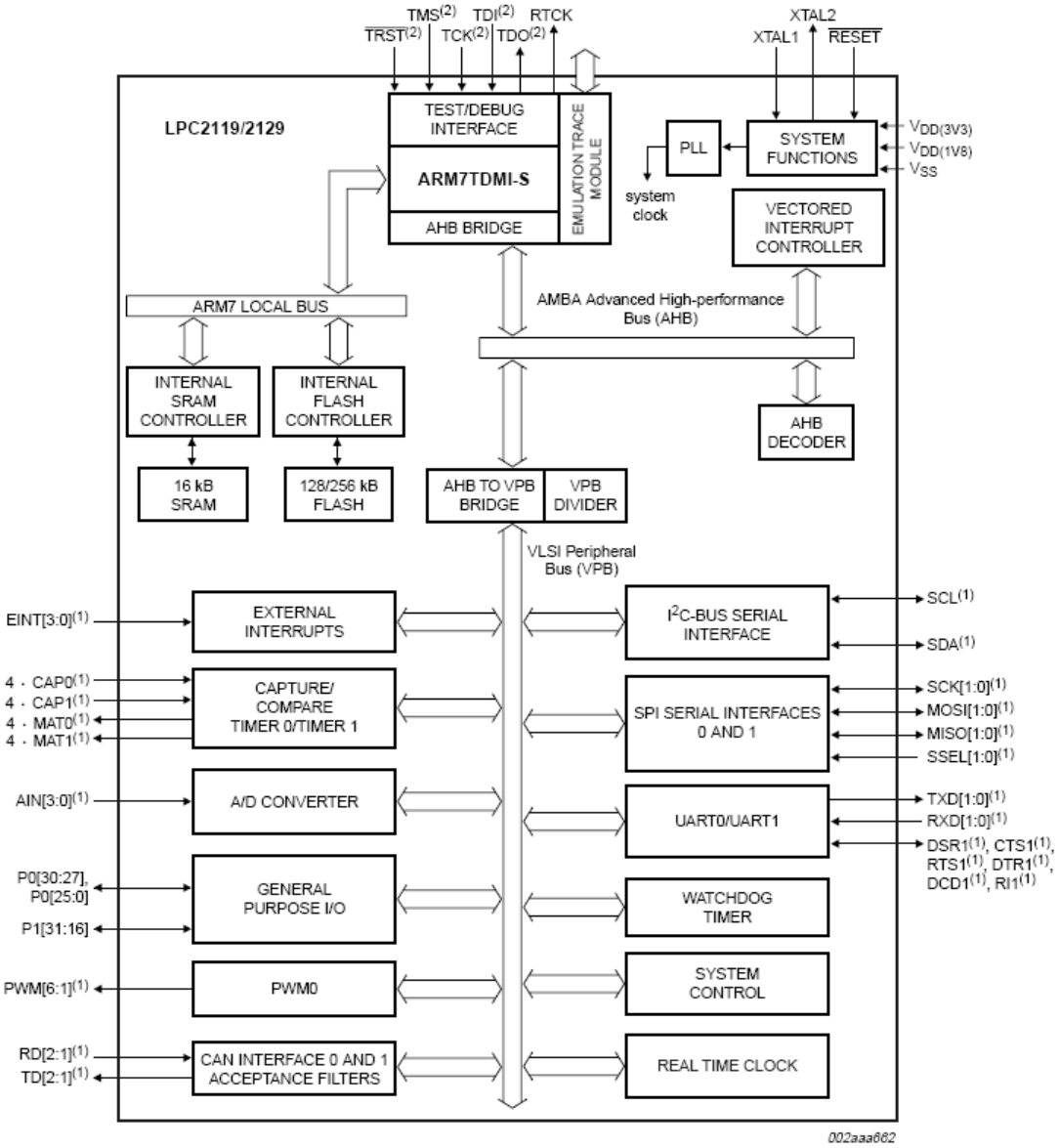
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
External clock						
f_{osc}	oscillator frequency	supplied by an external oscillator (signal generator)	1	-	50	MHz
		external clock frequency supplied by an external crystal oscillator	1	-	30	MHz
		external clock frequency if on-chip PLL is used	10	-	25	MHz
		external clock frequency if on-chip bootloader is used for initial code download	10	-	25	MHz
$T_{cy(clk)}$	clock cycle time		20	-	1000	ns
t_{CHCX}	clock HIGH time		$T_{cy(clk)} \times 0.4$	-	-	ns
t_{CLCX}	clock LOW time		$T_{cy(clk)} \times 0.4$	-	-	ns
t_{CLCH}	clock rise time		-	-	5	ns
t_{CHCL}	clock fall time		-	-	5	ns
Port pins (except P0.2 and P0.3)						
t_r	rise time		-	10	-	ns
t_f	fall time		-	10	-	ns
I^2C-bus pins (P0.2 and P0.3)						
t_f	fall time	V_{IH} to V_{IL}	[2] $20 + 0.1 \times C_b$	-	-	ns

Πίνακας 3: Χαρακτηριστικά δυναμικής λειτουργίας [4]



Εικόνα 7: Χρονισμός εξωτερικού ρολογιού [4]

Το μπλοκ διάγραμμά του δίνεται στην παρακάτω εικόνα.



(1) Shared with GPIO.

(2) When test/debug interface is used, GPIO/other functions sharing these pins are not available.

Εικόνα 8: Μπλοκ διάγραμμα του LPC2129 [4]

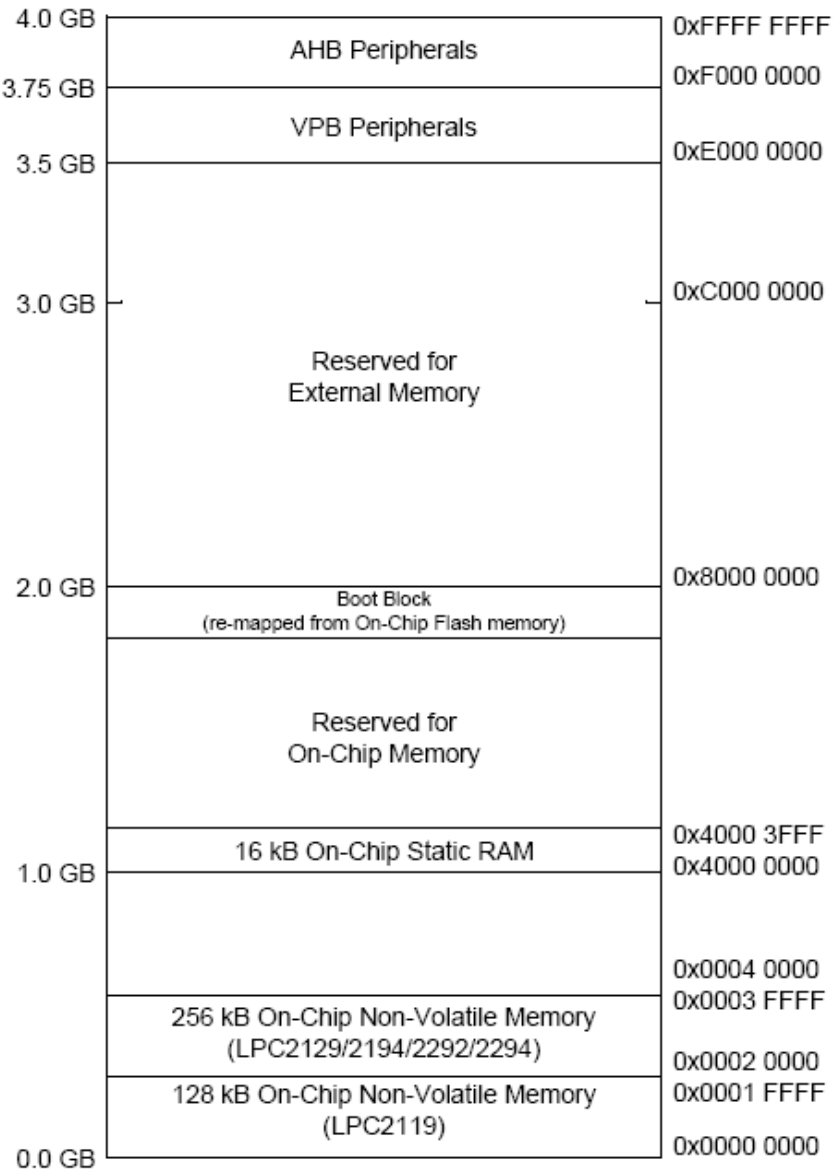
Τα παραπάνω χαρακτηριστικά διαμορφώνουν τον LPC2129 εξαιρετικά κατάλληλο για εφαρμογές βιομηχανικού ελέγχου, συστημάτων επικοινωνίας, ιατρικών συστημάτων κ.ά. αφού προσφέρει ένα πλήρες σύνολο δυνατοτήτων σε μικρό κόστος, υψηλές επιδόσεις, μικρή κατανάλωση και ελάχιστο χώρο.

Στη συνέχεια αναλύονται τα συστήματα εκείνα των οποίων γίνεται χρήση στα πλαίσια της εργασίας αυτής.

Σύστημα Μνήμης

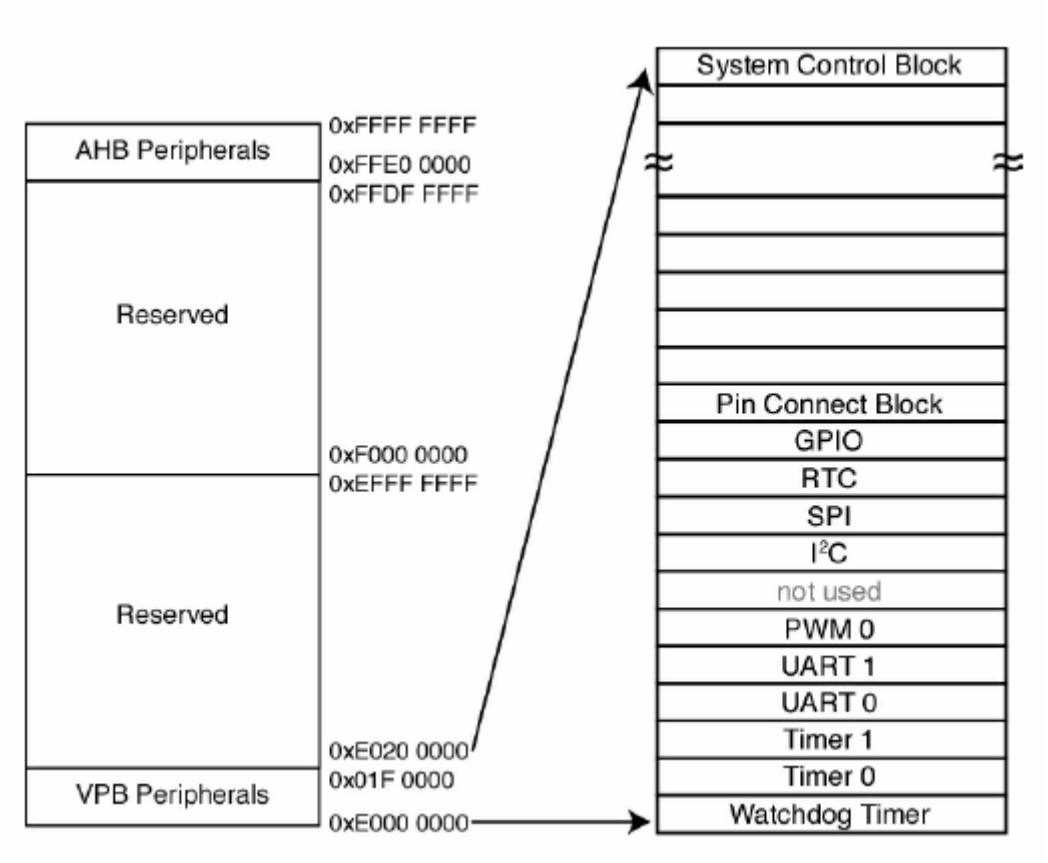
Χάρτης Μνήμης

Ο χάρτης μνήμης του LPC2129 είναι γραμμικός και περιλαμβάνει ξεχωριστές περιοχές μνήμης. Αυτές δείχνονται στην παρακάτω εικόνα και αποτελούν το σύνολο του χώρου μνήμης, όπως αυτός είναι διαθέσιμος στο πρόγραμμα του χρήστη ύστερα από μια επανεκκίνηση.



Εικόνα 9: Χάρτης μνήμης του συστήματος [5]

Διακρίνονται η περιοχή που ξεκινάει από τη θέση μνήμης 0x00000000 και είναι διαθέσιμη για την ενσωματωμένη μνήμη flash, τα περιεχόμενα της οποίας μένουν ανέπαφα μετά από μια επανεκκίνηση, η περιοχή που ξεκινάει από τη θέση 0x40000000 και δίνει πρόσβαση στην ενσωματωμένη στατική μνήμη RAM, τα περιεχόμενα της οποίας σβήνονται κατά την επανεκκίνηση, μια περιοχή μεταξύ 0x7FFFFFFF και 0x80000000 όπου βρίσκονται τα προγράμματα που απαιτούνται για την εκκίνηση, τον προγραμματισμό και την παρακολούθηση του μικροεπεξεργαστή και δύο περιοχές διαθέσιμες για πρόσβαση στους καταχωρητές των ενσωματωμένων περιφερειακών. Η περιοχή των περιφερειακών χρήστη, που είναι συνδεδεμένα στο δίαυλο περιφερειακών VLSI (VPB) αναλύεται ως εξής:



Εικόνα 10: Πρόσβαση στα περιφερειακά VLSI μέσω διευθύνσεων μνήμης [6]

Για κάθε περιφερειακό υπάρχει διαθέσιμος χώρος μνήμης μεγέθους 16kB. Οι θέσεις μνήμης που αντιστοιχούν στους διάφορους καταχωρητές των περιφερειακών αυτών είναι

ευθυγραμμισμένες σε λέξεις των 32 bit, κάτι που έχει ως αποτέλεσμα μικρότερες απαιτήσεις σε υλικό που θα ήταν αναγκαίο σε διαφορετική περίπτωση και το οποίο με τη σειρά του μεταφράζεται σε απλούστερη, ταχύτερη και πιο αξιόπιστη υλοποίηση. Ταυτόχρονα, αυτό σημαίνει ότι, σε επίπεδο προγράμματος, η πρόσβαση γίνεται σαν να ήταν όλοι οι καταχωρητές 32 bit, ανεξάρτητα του πραγματικού τους μεγέθους.

Στις πρώτες διευθύνσεις μνήμης είναι τοποθετημένα τα διανύσματα των διακοπών όπως φαίνεται στον παρακάτω πίνακα:

Address	Exception
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt
0x0000 000C	Prefetch Abort (instruction fetch memory fault)
0x0000 0010	Data Abort (data access memory fault)
0x0000 0014	Reserved *
0x0000 0018	IRQ
0x0000 001C	FIQ

Πίνακας 4: Θέσεις μνήμης των διανυσμάτων διακοπών [5]

Οι διευθύνσεις των διανυσμάτων διακοπών οφείλουν να βρίσκονται στην αρχή της περιοχής μνήμης στην οποία βρίσκεται και το πρόγραμμα που εκτελείται ώστε να μην απαιτείται αλλαγή του κώδικα ανάλογα με το σημείο στο οποίο αποθηκεύεται το πρόγραμμα. Για το λόγο αυτό, ενώ όταν το πρόγραμμα εκτελείται από την ενσωματωμένη μνήμη flash τα διανύσματα διακοπών βρίσκονται εκ κατασκευής στην αρχή της αντίστοιχης περιοχής μνήμης, το ίδιο παύει να ισχύει όταν το πρόγραμμα είναι τοποθετημένο στη μνήμη RAM. Στην περίπτωση αυτή, ο κατάλληλος καταχωρητής του συστήματος ελέγχου του επεξεργαστή ρυθμίζεται ώστε τα διανύσματα διακοπών να εμφανίζονται στην αρχή της μνήμης RAM και το πρόγραμμα να έχει πρόσβαση σε αυτά χωρίς καμία αλλαγή στον κώδικα. Το αντίστοιχο συμβαίνει όταν ο επεξεργαστής βρίσκεται σε κατάσταση εκτέλεσης του προγράμματος εκκίνησης (Boot loader), ο κώδικας του οποίου μεταφέρεται μαζί με τα διανύσματα διακοπών στο πάνω μέρος της περιοχής ενσωματωμένης μνήμης.

Memory Accelerator Module

Ένα από τα καθοριστικά υποσυστήματα του συστήματος μνήμης του LPC2129 είναι η MAM (Memory Accelerator Module – Μονάδα Επιτάχυνσης Μνήμης). Η μονάδα αυτή συνδέεται στον τοπικό δίαυλο (local bus) του επεξεργαστή και παρεμβάλλεται μεταξύ της ΚΜΕ και της μνήμης flash. Προορίζεται για την αντιμετώπιση της καθυστέρησης που θα υπήρχε αν η ΚΜΕ είχε άμεση πρόσβαση στη μνήμη flash, καθώς η δεύτερη είναι έως τρεις φορές πιο αργή από την ταχύτητα της ΚΜΕ.

Η λειτουργία της μονάδας είναι αντίστοιχη με τη λειτουργία μιας μνήμης cache, υπό την έννοια ότι αναλαμβάνει να προετοιμάσει τις εντολές που πρόκειται να εκτελεστούν από την ΚΜΕ, φέρνοντάς τις από την κεντρική μνήμη, η οποία στη συγκεκριμένη περίπτωση είναι η μνήμη flash. Επειδή όμως η ύπαρξη μιας πλήρους μνήμης cache θα απαιτούσε σημαντική περιοχή του chip του LPC2129 και ένα μεγάλο αριθμό πυλών, ερχόμενη σε αντίθεση με τη φιλοσοφία σχεδίασής του, που βασίζεται στην απλότητα του υλικού, η MAM είναι ένα σαφώς απλούστερο κύκλωμα που φροντίζει να διαβάζει από τη μνήμη και να διαθέτει στον επεξεργαστή τις επόμενες 4 εντολές που προβλέπεται να εκτελεστούν. Αυτό επιτυγχάνεται με την ύπαρξη δύο buffers των 128 bit οι οποίοι είναι διαδοχικά διαθέσιμοι στην ΚΜΕ και περιέχουν 4 εντολές ARM των 32 bit ή 8 εντολές THUMB των 16 bit που έχουν φέρει από τη μνήμη flash. Εκτός της απλής λειτουργίας που περιγράφηκε, η μονάδα επιτάχυνσης της μνήμης περιλαμβάνει και κάποια απλά υποσυστήματα για την πρόβλεψη περιπτώσεων μη γραμμικής εκτέλεσης εντολών σε μικρούς βρόχους, ώστε να αυξάνεται η επίδοσή της χωρίς να αυξάνεται σημαντικά η πολυπλοκότητά της.

Η MAM ελέγχεται από δύο καταχωρητές: έναν χρονισμού και έναν ελέγχου, ώστε να μπορεί να τίθεται σε μερική ή πλήρη λειτουργία και όταν το επιθυμεί ο χρήστης. Επίσης παρέχει σε καταχωρητές κάποια στατιστικά στοιχεία σχετικά με τις επιδόσεις της ώστε να είναι δυνατή η βελτιστοποίηση του κώδικα ανάλογα με τις ανάγκες του χρήστη. Συνολικά, η ενεργοποίηση της MAM έχει κατά μέσο όρο σαν αποτέλεσμα μια σημαντική βελτίωση της ταχύτητας εκτέλεσης ενός προγράμματος χωρίς βέβαια να επιτυγχάνονται επιδόσεις αντίστοιχες με αυτές μιας πλήρους μνήμης cache αλλά με περισσότερο ντετερμινιστική συμπεριφορά της εκτέλεσης.

Σύστημα Ελέγχου

Γενικά

Το Σύστημα Ελέγχου περιλαμβάνει τους καταχωρητές που ελέγχουν γενικές λειτουργίες του μικροεπεξεργαστή που δεν σχετίζονται με κάποιο συγκεκριμένο περιφερειακό. Αυτές είναι:

- Ο κρυσταλλικός ταλαντωτής
- Οι είσοδοι εξωτερικών διακοπών
- Το σύστημα χαρτογράφησης μνήμης
- Ο PLL (Phase Locked Loop)
- Η διαχείριση ενέργειας
- Η επανεκκίνηση
- Ο διαιρέτης VPB (VLSI Peripheral Bus)
- Ο χρονομετρητής αφύπνισης

Οι αντίστοιχοι καταχωρητές βρίσκονται συγκεντρωμένοι στον παρακάτω πίνακα ενώ οι λεπτομέρειές τους αναλύονται στη συνέχεια:

Name	Description	Access	Reset Value*	Address
External Interrupts				
EXTINT	External Interrupt Flag Register.	R/W	0	0xE01FC140
EXTWAKE	External Interrupt Wakeup Register.	R/W	0	0xE01FC144
EXTMODE	External Interrupt Mode Register.	R/W	0	0xE01FC148
EXTPOLAR	External Interrupt Polarity Register.	R/W	0	0xE01FC14C
Memory Mapping Control				
MEMMAP	Memory Mapping Control.	R/W	0	0xE01FC040
Phase Locked Loop				
PLLCON	PLL Control Register.	R/W	0	0xE01FC080
PLLCFG	PLL Configuration Register.	R/W	0	0xE01FC084
PLLSTAT	PLL Status Register.	RO	0	0xE01FC088
PLLFEED	PLL Feed Register.	WO	NA	0xE01FC08C
Power Control				
PCON	Power Control Register.	R/W	0	0xE01FC0C0
PCONP	Power Control for Peripherals.	R/W	0x3BE	0xE01FC0C4
VPB Divider				
VPBDIV	VPB Divider Control.	R/W	0	0xE01FC100

*Η τιμή μετά την επανεκκίνηση (Reset) αναφέρεται μόνο στα bits που χρησιμοποιούνται. Η τιμή των αχρησιμοποίητων (reserved) bits είναι ακαθόριστη.

Πίνακας 5: Καταχωρητές Συστήματος Ελέγχου [5]

Κρυσταλλικός ταλαντωτής

Ο κρυσταλλικός ταλαντωτής του μικροεπεξεργαστή μπορεί να συνδεθεί (στους ακροδέκτες X1 και X2) με εξωτερικό κρύσταλλο στο εύρος του 1Mhz έως 30Mhz. Αν όμως χρησιμοποιείται ο PLL, τότε το εύρος αυτό μειώνεται στα 10Mhz έως 25Mhz. Η συχνότητα του ταλαντωτή, που είναι ίση με τη συχνότητα του κρυστάλλου, θα ονομάζεται F_{osc} ενώ η συχνότητα του ρολογιού της ΚΜΕ θα ονομάζεται cclk (Computer Clock). Η σχέση μεταξύ των δύο αυτών τιμών εξαρτάται από τις ρυθμίσεις του PLL, όπως θα δειχθεί στη συνέχεια. Η τυπωμένη πλακέτα περιλαμβάνει κρύσταλλο που ταλαντώνεται στα 12MHz και ο PLL ρυθμίζεται έτσι ώστε η συχνότητα της ΚΜΕ να είναι στα 60MHz.

Είσοδοι εξωτερικών διακοπών

Ο LPC2129 υποστηρίζει 4 εισόδους εξωτερικών διακοπών ως επιλεγόμενες λειτουργίες ακροδεκτών. Αυτές αναφέρονται ως EINT# (External Interrupt No.#) και μπορούν να αντιστοιχιστούν στους παρακάτω ακροδέκτες:

Όνομα Ακροδέκτη	Αριθμός Ακροδέκτη
EINT0	P0.1 ή P0.16
EINT1	P0.3 ή P0.14
EINT2	P0.7 ή P0.15
EINT3	P0.9 ή P0.20 ή P0.30

Πίνακας 6: Εξωτερικές διακοπές και ακροδέκτες [5]

Αν ένας από τους ακροδέκτες της εξωτερικής διακοπής EINT0 έχει δυναμικό 0 μετά το τέλος της επανεκκίνησης, τότε ενεργοποιείται η λειτουργία ISP (In-System Programming) που χρησιμοποιείται για τον προγραμματισμό του μικροεπεξεργαστή μέσω της σειριακής θύρας.

Οι εξωτερικές διακοπές μπορούν να ρυθμιστούν ξεχωριστά ώστε να είναι ευαίσθητες σε στάθμη ή σε ακμή δυναμικού στους ακροδέκτες. Επίσης, επιλέγεται αν είναι ενεργές σε υψηλό ή χαμηλό δυναμικό ή αντίστοιχα σε θετική ή αρνητική ακμή. Τέλος, υπάρχει η δυνατότητα επιλογής για την αφύπνιση του μικροεπεξεργαστή από λειτουργία εξοικονόμησης ενέργειας μέσω των διακοπών αυτών.

Ο καταχωρητής **EXTINT** περιέχει την πληροφορία για το αν κάποια από τις εξωτερικές διακοπές είναι ενεργή. Τα bits 0, 1, 2 και 3 είναι μονάδες όταν η αντίστοιχη διακοπή (EINT0, EINT1, EINT2 και EINT3) είναι ενεργή, ενώ τα υπόλοιπα bits δεν χρησιμοποιούνται. Η εγγραφή μονάδας σε κάποιο από τα bits αυτά τα κάνει μηδενικά, εφόσον η πηγή της διακοπής δεν είναι πλέον ενεργή, και χρησιμοποιείται για την επιβεβαίωση λήψης της διακοπής.

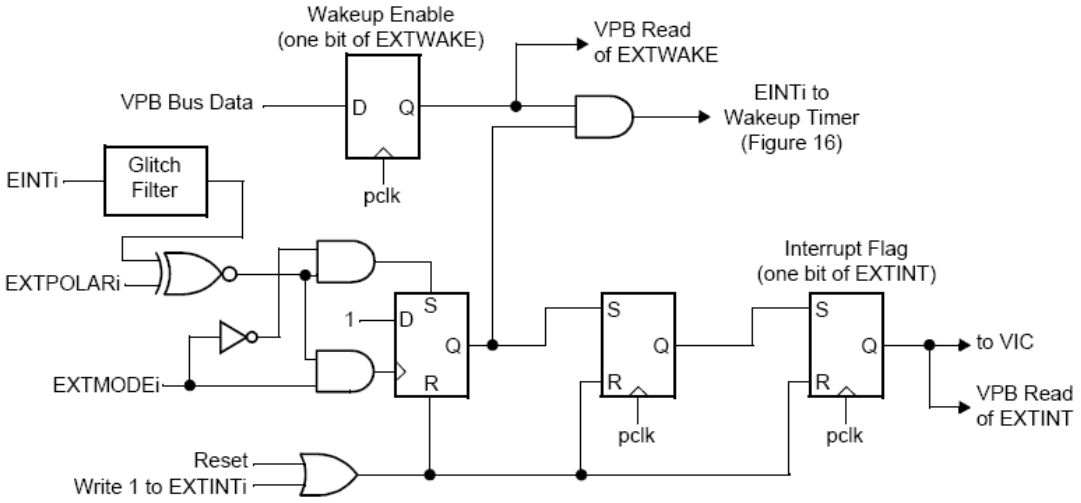
Ο καταχωρητής **EXTWAKE** καθορίζει ποιες από τις εξωτερικές διακοπές προκαλούν αφύπνιση του μικροεπεξεργαστή. Όταν το αντίστοιχο bit είναι μονάδα, μια ενεργοποίηση της διακοπής θα προκαλέσει αφύπνιση ακόμα και αν στο σύστημα διακοπών δεν έχει οριστεί η εκτέλεση κάποιας ρουτίνας εξυπηρέτησης της διακοπής αυτής.

Τα τέσσερα χαμηλότερα bits του καταχωρητή **EXTMODE** ορίζουν αν κάθε διακοπή ενεργοποιείται από στάθμη ή από ακμή του δυναμικού του αντίστοιχου pin. Μονάδα σημαίνει ότι η διακοπή ενεργοποιείται από στάθμη ενώ μηδενικό από ακμή.

Καθένα από τα τέσσερα χαμηλότερα bits του καταχωρητή **EXTPOLAR** καθορίζει, όταν είναι μονάδα, ότι η αντίστοιχη διακοπή είναι ενεργή σε υψηλό δυναμικό ή σε θετική ακμή ενώ, όταν είναι μηδενικό, ότι η αντίστοιχη διακοπή είναι ενεργή σε χαμηλό δυναμικό ή σε αρνητική ακμή, ανάλογα με τη ρύθμιση που έχει γίνει στον καταχωρητή EXTMODE.

Οι τιμές των καταχωρητών μετά από μια επανεκκίνηση του μικροεπεξεργαστή είναι μηδενικές ώστε όλες οι εξωτερικές διακοπές να είναι ρυθμισμένες ως ενεργές σε χαμηλή στάθμη χωρίς να προκαλούν αφύπνιση του συστήματος. Οι όποιες αλλαγές στους καταχωρητές αυτούς πρέπει να γίνονται με τις διακοπές απενεργοποιημένες στο πρόγραμμα του χρήστη.

Αν ο προγραμματιστής έχει ορίσει δύο ακροδέκτες για κάποια από τις εξωτερικές διακοπές, τότε η διακοπή ενεργοποιείται όταν ένας τουλάχιστον από τους δύο ακροδέκτες ικανοποιεί τις συνθήκες που έχουν οριστεί ώστε η διακοπή να είναι ενεργή. Ο ορισμός δύο ακροδεκτών για εξωτερική διακοπή που έχει ρυθμιστεί να ενεργοποιείται σε ακμές θεωρείται προγραμματιστικό σφάλμα. Στην περίπτωση αυτή πάντως, υπερισχύει η κατάσταση του ακροδέκτη με το χαμηλότερο αριθμό. Το κύκλωμα αναγνώρισης εξωτερικής διακοπής είναι το παρακάτω:



Εικόνα 11: Κύκλωμα αναγνώρισης εξωτερικής διακοπής [5]

Το σύστημα διαχείρισης διακοπών του LPC2129 είναι εξαιρετικά ανεπτυγμένο προσφέροντας πληθώρα δυνατοτήτων αλλά δεν θα χρησιμοποιηθεί άμεσα στη συγκεκριμένη εφαρμογή, γι' αυτό και δεν θα αναλυθεί περισσότερο.

Σύστημα χαρτογράφησης μνήμης

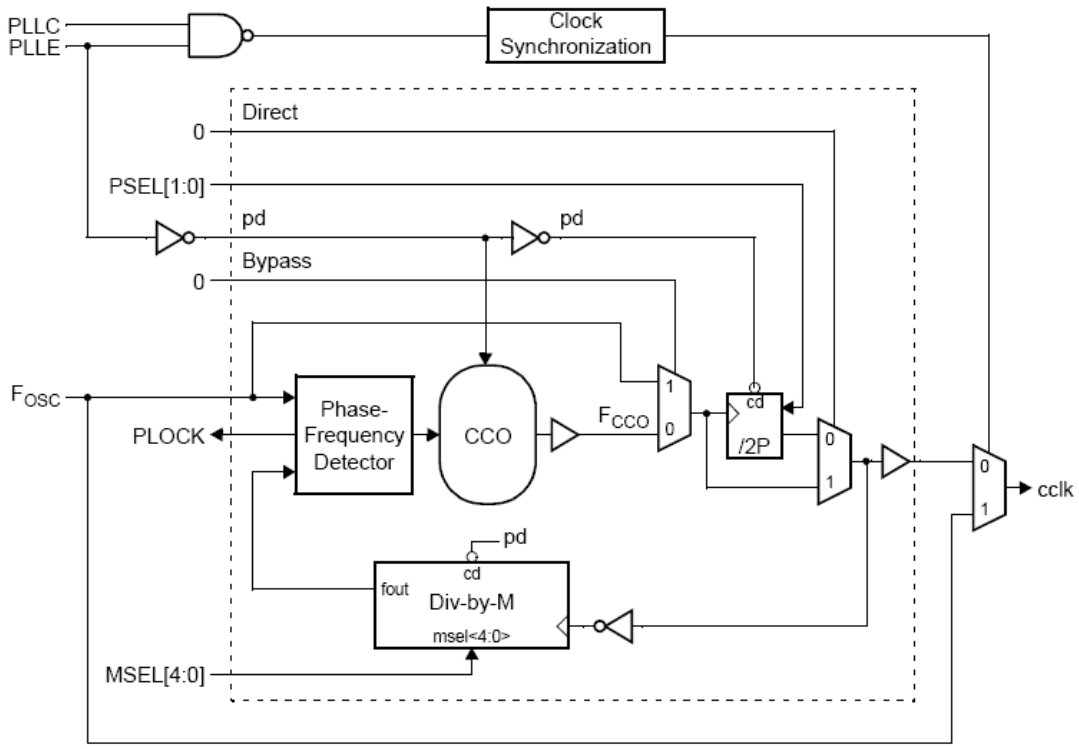
Το σύστημα χαρτογράφησης μνήμης καθορίζει την πραγματική θέση των διανυσμάτων διακοπών στη μνήμη. Τα διανύσματα αυτά καθορίζουν τις εντολές που εκτελούνται όταν προκληθεί η αντίστοιχη διακοπή και για τον προγραμματιστή εμφανίζονται να βρίσκονται στη διεύθυνση 0x00000000. Η τοποθέτησή τους σε διαφορετικές περιοχές στη μνήμη τα καθιστά προσβάσιμα στον κώδικα που τρέχει στην περιοχή αυτή.

Μόνο τα δύο LSB του καταχωρητή του συστήματος χαρτογράφησης μνήμης **MEMMAP** χρησιμοποιούνται και:

- Η τιμή 00 ορίζει τη λειτουργία Boot Loader, οπότε τα διανύσματα διακοπών μεταφέρονται στο Boot Block
- Η τιμή 01 ορίζει τη λειτουργία User Flash, οπότε τα διανύσματα διακοπών δεν μεταφέρονται και παραμένουν στη μνήμη Flash
- Η τιμή 10 ορίζει τη λειτουργία User RAM, οπότε τα διανύσματα μεταφέρονται στην στατική μνήμη RAM

PLL (Phase Locked Loop)

Ο PLL δέχεται μια συχνότητα εισόδου από τον κρυσταλλικό ταλαντωτή F_{osc} στο εύρος 10 έως 25Mhz. Η συχνότητα εξόδου, που αποτελεί και τη συχνότητα του ρολογιού του επεξεργαστή cclk, είναι η συχνότητα εισόδου αφού πολλαπλασιαστεί με μια ακέραιη τιμή μεταξύ 1 και 32 μέσω ενός ταλαντωτή ελεγχόμενου από ρεύμα (Current Controlled Oscillator – CCO). Η συχνότητα του CCO πρέπει να βρίσκεται μεταξύ 156 και 320MHz. Για το λόγο αυτό, στο κύκλωμα υπάρχει και ένας διαιρέτης που παίρνει τιμές 2, 4, 8 ή 16. Η συχνότητα εξόδου πρέπει να είναι μεταξύ 10 και 60Mhz. Το κύκλωμα του PLL είναι το εξής:



Εικόνα 12: Κύκλωμα PLL [5]

Οι ρυθμίσεις και η κατάσταση του PLL βρίσκονται στους αντίστοιχους καταχωρητές. Εξαιτίας της σημασίας που έχει ο PLL για τη σωστή λειτουργία του συστήματος, οι μεταβολές στους καταχωρητές του προστατεύονται με την απαίτηση ύπαρξης κατάλληλης ακολουθίας εντολών στο πρόγραμμα. Ο PLL μετά την επανεκκίνηση του

συστήματος ή κατά την είσοδο σε κατάσταση μη-λειτουργίας (βλ. *Διαχείριση ενέργειας*) είναι απενεργοποιημένος, ενώ σε κατάσταση αδράνειας παραμένει ενεργός.

Το bit 0 του καταχωρητή **PLLCON** ονομάζεται PLLE (Enable) και όταν είναι μονάδα, ο PLL ενεργοποιείται και προσπαθεί να κλειδώσει στην επιθυμητή συχνότητα. Το bit 1 ονομάζεται PLLC (Connect) και όταν είναι μονάδα ορίζει το ρολόι της ΚΜΕ cclk ίσο με την έξοδο του PLL. Τα υπόλοιπα bits δεν χρησιμοποιούνται.

Τα 5 χαμηλότερα bits του καταχωρητή **PLLCFG** καθορίζουν την τιμή M με την οποία πολλαπλασιάζεται η συχνότητα είσοδου. Τα επόμενα 2 καθορίζουν την τιμή P με το διπλάσιο της οποίας διαιρείται η προηγούμενη για να προκύψει η έξοδος του PLL. Το υψηλότερο bit δεν χρησιμοποιείται. Οι τιμές του M και του P δίνονται από τους παρακάτω πίνακες:

MSEL Bits (PLLCFG bits 4:0)	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

PSEL Bits (PLLCFG bits 6:5)	Value of P
00	1
01	2
10	4
11	8

Πίνακας 7: Τιμές των M και P του PLL [5]

Η συχνότητα εξόδου του PLL δίνεται από τη σχέση:

$$cclk = M \cdot F_{osc}$$

και η συχνότητα του CCO από τη σχέση:

$$F_{cco} = F_{osc} \cdot M \cdot 2 \cdot P$$

Ο καταχωρητής **PLLSTAT** είναι μόνο για ανάγνωση, περιέχει πληροφορίες σχετικά με την τρέχουσα κατάσταση του PLL και μπορεί να διαφέρει από τις ρυθμίσεις που έχουν γίνει στους προηγούμενους καταχωρητές, καθώς οι τελευταίες

πραγματοποιούνται μόνο αφού υπάρξει η κατάλληλη ακολουθία εντολών στο πρόγραμμα. Τα bits 4:0 περιέχουν την τιμή του πολλοπλασιαστή. Τα bits 6:5 περιέχουν την τιμή του διαιρέτη. Το bit 8 περιέχει την τρέχουσα τιμή του PPLE και το bit 9 του PPLC. Το bit 10, που ονομάζεται PLOCK, είναι μονάδα όταν ο PLL έχει κλειδώσει στην επιθυμητή συχνότητα και μηδενικό διαφορετικά. Τα υπόλοιπα bits δεν χρησιμοποιούνται.

Ο καταχωρητής **PLLFEED** χρησιμοποιείται μόνο για την ενεργοποίηση των ρυθμίσεων που έχουν οριστεί στους προηγούμενους καταχωρητές. Η ενεργοποίηση γίνεται με διαδοχική εγγραφή στον PLLFEED των τιμών 0xAA και 0x55. Προκειμένου οι δύο αυτές εντολές εγγραφής να εκτελεστούν διαδοχικά, θα πρέπει να έχουν απενεργοποιηθεί οι διακοπές.

Η διαδικασία ενεργοποίησης του PLL ακολουθεί τα παρακάτω βήματα:

- Υπολογισμός των τιμών M και P
- Ρύθμιση του καταχωρητή PLLCFG και ενεργοποίηση του PLLE
- Διαδοχική εγγραφή των τιμών 0xAA και 0x55 στον PLLFEED
- Αναμονή μέχρι το PLOCK να γίνει μονάδα
- Ενεργοποίηση του PPLC
- Διαδοχική εγγραφή των τιμών 0xAA και 0x55 στον PLLFEED
- Συνέχιση του προγράμματος

Το σήμα PLOCK είναι συνδεδεμένο με το σύστημα διακοπών και προκαλεί διακοπή όταν το κλείδωμα του PLL στην επιθυμητή συχνότητα επιτευχθεί. Έτσι, το πρόγραμμα μπορεί να συνεχίσει να εκτελείται έως ότου το PLOCK γίνει μονάδα χωρίς την ανάγκη διαρκούς ανάγνωσής του. Η ρουτίνα εξυπηρέτησης της διακοπής θα φροντίσει να ενεργοποιήσει το PPLC και να ορίσει το cclk ίσο με την έξοδο του PLL.

Διαχείριση ενέργειας

Ο LPC2129 διαθέτει δύο καταστάσεις λειτουργίας εξοικονόμησης ενέργειας στις οποίες μπορεί να εισέλθει με κατάλληλες εντολών: την κατάσταση αδράνειας (Idle mode) και την κατάσταση μη-λειτουργίας (Power Down mode).

Στην πρώτη, η εκτέλεση των εντολών σταματάει ενώ η λειτουργία των περιφερειακών συνεχίζεται κανονικά, με αποτέλεσμα να εξαλείφεται η κατανάλωση ενέργειας από την ΚΜΕ, το σύστημα μνήμης και τους εσωτερικούς διαύλους. Ο μικροεπεξεργαστής μπορεί να επανέλθει σε κανονική λειτουργία όταν συμβεί μια διακοπή (είτε εξωτερική είτε από κάποιο περιφερειακό) ή ύστερα από επανεκκίνηση.

Στη δεύτερη, κάθε δυναμική λειτουργία του συστήματος διακόπτεται καθώς το ρολόι παύει να λειτουργεί, οπότε η κατανάλωση ενέργειας προσεγγίζει το μηδέν. Η κατάσταση του μικροεπεξεργαστή παραμένει ως είχε όταν αυτός εισήλθε στην κατάσταση μη-λειτουργίας και επανέρχεται σε κανονική λειτουργία ύστερα από διακοπή που μπορεί να λειτουργήσει χωρίς ρολόι (δηλαδή εξωτερική ή διακοπή από το υποσύστημα CAN) ή με επανεκκίνηση.

Σε κάθε περίπτωση, η είσοδος και έξοδος από καταστάσεις εξοικονόμησης ενέργειας αφήνει τα δεδομένα του μικροπεξεργαστή ανέπαφα ώστε η εκτέλεση να συνεχίσει χωρίς προβλήματα. Επιπλέον, το σύστημα διαχείρισης ενέργειας επιτρέπει στον προγραμματιστή να απενεργοποιήσει σε κανονική λειτουργία τα περιφερειακά που δεν χρειάζεται ώστε να μην υπάρχουν οι αντίστοιχες καταναλώσεις. Μετά από κάθε επανεκκίνηση, όλα τα περιφερειακά είναι ενεργά.

Ο καταχωρητής **PCON** περιέχει δύο bits που αντιστοιχούν στις δύο καταστάσεις εξοικονόμησης ενέργειας. Όταν το bit 0 οριστεί μονάδα, ο επεξεργαστής μπαίνει σε κατάσταση αδράνειας (Idle mode) ενώ όταν το bit 1 οριστεί μονάδα, ο επεξεργαστής μπαίνει σε κατάσταση μη-λειτουργίας (Power Down mode). Ύστερα από την επαναφορά σε κατάσταση κανονικής λειτουργίας, τα bits αυτά γίνονται 0. Τα υπόλοιπα bits δεν χρησιμοποιούνται.

Ο καταχωρητής **PCONP** ελέγχει τη λειτουργία ή την απενεργοποίηση των διαφόρων περιφερειακών. Κάθε bit αντιστοιχεί σε ένα συγκεκριμένο περιφερειακό του LPC2129, το οποίο και απενεργοποιείται όταν οριστεί μηδέν. Η αντιστοίχιση φαίνεται στον παρακάτω πίνακα:

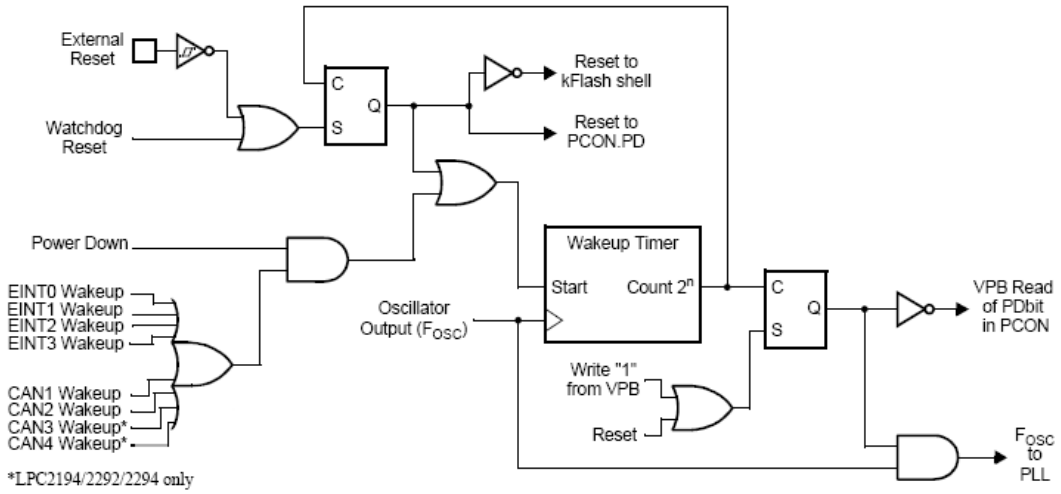
PCONP	Function	Description	Reset Value
0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
1	PCTIM0	When 1, TIMER0 is enabled. When 0, TIMER0 is disabled to conserve power.	1
2	PCTIM1	When 1, TIMER1 is enabled. When 0, TIMER1 is disabled to conserve power.	1
3	PCURT0	When 1, UART0 is enabled. When 0, UART0 is disabled to conserve power.	1
4	PCURT1	When 1, UART1 is enabled. When 0, UART1 is disabled to conserve power.	1
5	PCPWM0	When 1, PWM0 is enabled. When 0, PWM0 is disabled to conserve power.	1
6	Reserved	User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	0
7	PCI2C	When 1, the I^2C interface is enabled. When 0, the I^2C interface is disabled to conserve power.	1
8	PCSPI0	When 1, the SPI0 interface is enabled. When 0, the SPI0 is disabled to conserve power.	1
9	PCRTC	When 1, the RTC is enabled. When 0, the RTC is disabled to conserve power.	1
10	PCSPI1	When 1, the SPI1 interface is enabled. When 0, the SPI1 is disabled to conserve power.	1
11	Reserved	User software should write 0 here to reduce power consumption.	1
12	PCAD	When 1, the A/D converter is enabled. When 0, the A/D is disabled to conserve power.	1
13	PCCAN1	When 1, CAN Controller 1 is enabled. When 0, it is disabled to save power. **Note:** the Acceptance Filter is enabled if any of CAN Controllers 1-2 is enabled.	1
14	PCCAN2	When 1, CAN Controller 2 is enabled. When 0, it is disabled to save power.	1
31:15	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Πίνακας 8: Καταχωρητής ελέγχου λειτουργίας περιφερειακών PCONP [5]

Επανεκκίνηση

Η επανεκκίνηση του μικροεπεξεργαστή μπορεί να γίνει είτε εξωτερικά, κρατώντας τον ακροδέκτη $\overline{RESET}$ σε χαμηλό δυναμικό, είτε εσωτερικά από το Watchdog, σύστημα που ελέγχει τη σωστή λειτουργία της ΚΜΕ. Για να πραγματοποιηθεί εξωτερική επανεκκίνηση πρέπει ο αντίστοιχος ακτοδέκτης να παραμείνει σε χαμηλό δυναμικό μέχρι τη σταθεροποίηση του κρυσταλλικού ταλαντωτή. Αυτό απαιτεί περίπου 10ms όταν το σύστημα τροφοδοτείται με τάση για πρώτη φορά και περίπου 300ns όταν το σύστημα βρίσκεται ήδη σε λειτουργία.

Η επιβεβαίωση του σήματος επανεκκίνησης θέτει σε λειτουργία το χρονομετρητή αφύπνισης (βλ. *Χρονομετρητής αφύπνισης*) ο οποίος ξεκινάει με ορισμένη καθυστέρηση την αρχικοποίηση της μνήμης flash και όλων των καταχωρητών, ελέγχει την κατάσταση κάποιων ειδικών ακροδεκτών και, εφόσον δεν υπάρχει τρέχουσα διαδικασία εγγραφής της μνήμης flash, ολοκληρώνει την επανεκκίνηση ξεκινώντας την εκτέλεση των εντολών από τη διεύθυνση 0. Το κύκλωμα επανεκκίνησης είναι το εξής:



Εικόνα 13: Κύκλωμα επανεκκίνησης [5]

Διαιρέτης VPB

Ο διαιρέτης VPB είναι υπεύθυνος για τον έλεγχο της συχνότητας του ρολογιού (pclk) που δέχονται τα περιφερειακά που είναι συνδεδεμένα στο δίαυλο VPB. Η τιμή των δύο χαμηλότερων bits του καταχωρητή **VPBDIV** καθορίζουν την τιμή με την οποία διαιρείται το ρολόι του μικροεπεξεργαστή (cclk) ώστε να προκύψει το ρολόι του διαύλου VPB ως εξής:

- Η τιμή 00 διαιρεί το cclk με το τέσσερα
- Η τιμή 01 θέτει το pclk ίσο με το cclk
- Η τιμή 10 διαιρεί το cclk με το δύο
- Η τιμή 11 δεν χρησιμοποιείται

Τα υπόλοιπα bits δεν χρησιμοποιούνται.

Κατά την επανεκκίνηση του συστήματος, ο VPB είναι ρυθμισμένος έτσι ώστε το pclk να είναι το ¼ του cclk. Η ρύθμιση αυτή είναι η πλέον ασφαλής, καθώς επιτρέπει και σε πιο αργά περιφερειακά να λειτουργούν. Η τιμή του VPB καθορίζεται από τον προγραμματιστή έτσι ώστε τα περιφερειακά για τα οποία απαιτείται μεγάλη ταχύτητα να έχουν ρολόι ίσο με το cclk ενώ όταν η ταχύτητά τους δεν είναι κρίσιμη, να μπορούν να λειτουργούν σε μικρότερη συχνότητα καταναλώνοντας μικρότερη ενέργεια.

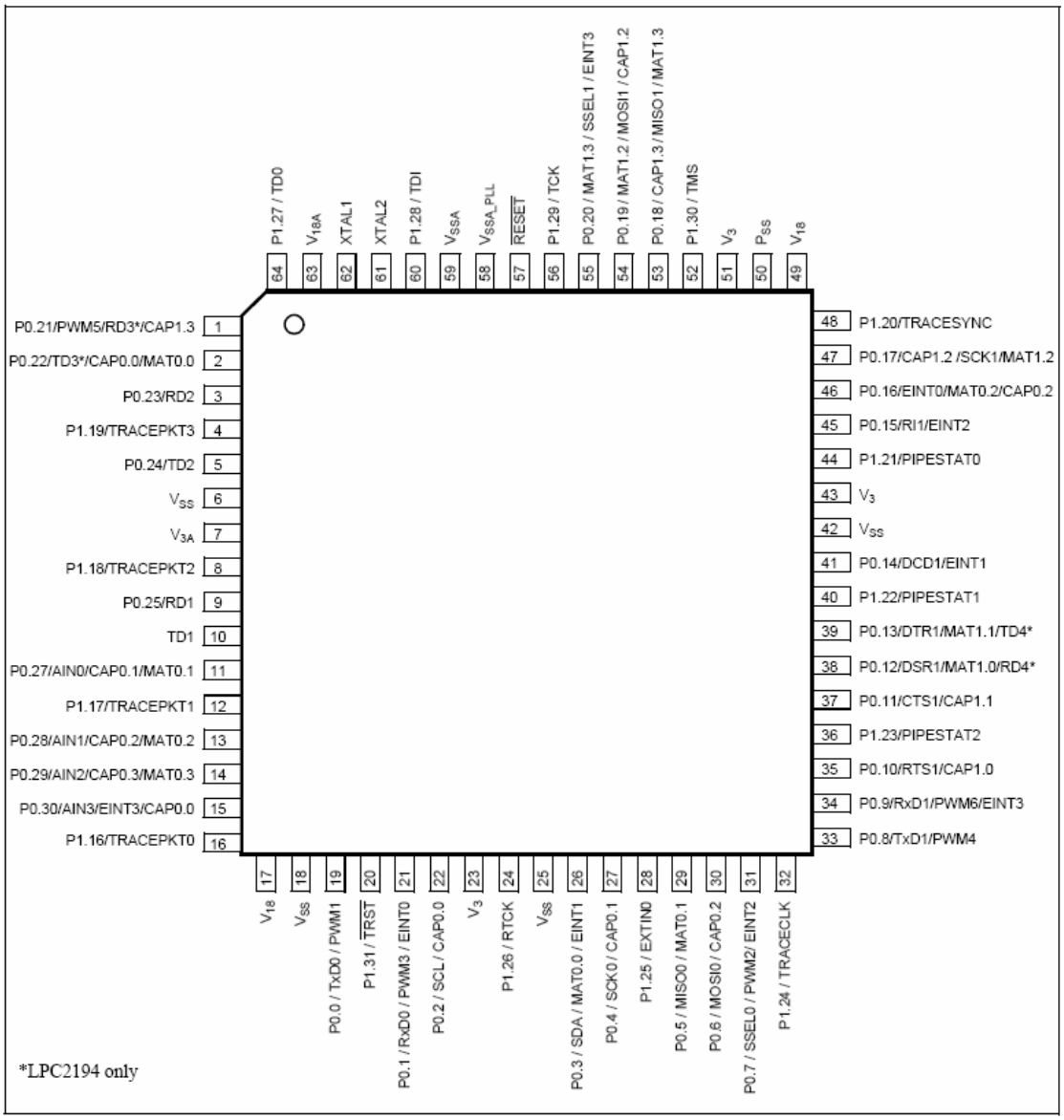
Χρονομετρητής αφύπνισης

Ο χρονομετρητής αφύπνισης είναι μια ενσωματωμένη συσκευή που τίθεται σε λειτουργία όταν υπάρξει σήμα επανεκκίνησης ή εξόδου από κατάσταση μη-λειτουργίας. Ελέγχει τη σωστή λειτουργία του κρυσταλλικού ταλαντωτή και όταν αυτή επιτευχθεί, μετράει 4096 παλμούς ρολογιού και ξεκινάει την αρχικοποίηση της μνήμης flash. Όταν αυτή ολοκληρωθεί, επιτρέπει στην KME να ξεκινήσει την εκτέλεση των εντολών από τη διεύθυνση 0.

Η εκκίνηση του χρονομετρητή αφύπνισης συμβαίνει όπως αναφέρθηκε είτε από σήμα επανεκκίνησης είτε από σήμα εξόδου από κατάσταση μη-λειτουργίας, δηλαδή από εξωτερική διακοπή ή διακοπή που προκαλείται από το υποσύστημα CAN. Εξαιτίας όμως της χρήσης των ακροδεκτών εξωτερικών διακοπών και ως ακροδεκτών συγκεκριμένων περιφερειακών, το σύστημα μπορεί να ρυθμιστεί έτσι ώστε να βγαίνει από κατάσταση μη-λειτουργίας όταν υπάρξει δραστηριότητα σε κάποιο από αυτά τα περιφερειακά. Για την υλοποίηση του σχήματος αυτού, ο προγραμματιστής πρέπει να φροντίσει πριν την είσοδο σε κατάσταση μη-λειτουργίας να ορίσει τη χρήση των ακροδεκτών από σήματα περιφερειακών σε σήματα εξωτερικών διακοπών. Όταν εμφανιστεί δραστηριότητα (συνήθως ένα χαμηλό δυναμικό) στους ακροδέκτες αυτούς λόγω διασύνδεσης του περιφερειακού με το περιβάλλον, ο μικροεπεξεργαστής θα επανέλθει σε κατάσταση κανονικής λειτουργίας, αφού όμως περάσει το χρονικό διάστημα που απαιτείται από το χρονομετρητή αφύπνισης. Η καθυστέρηση αυτή μπορεί να προκαλέσει απώλεια της δραστηριότητας που προκάλεσε την αφύπνιση, καθώς το περιφερειακό βρισκόταν εκτός λειτουργίας, οπότε το αντίστοιχο δεδομένο να χαθεί. Έτσι λοιπόν, όταν το δεδομένο αυτό είναι σημαντικό, ο προγραμματιστής θα πρέπει να χρησιμοποιήσει κατάσταση αδράνειας και όχι μη-λειτουργίας, στην οποία κατάσταση αδράνειας η αφύπνιση προκαλείται από το περιφερειακό αυτό καθεαυτό που, ώντας σε λειτουργία, φροντίζει να αναγνωρίσει το περιεχόμενο της δραστηριότητας.

Ρύθμιση Ακροδεκτών

Οι 64 ακροδέκτες του LPC2129 έχουν πολλαπλές χρήσεις, η επιλογή μεταξύ των οποίων γίνεται με τη χρήση τριών καταχωρητών. Η θέση και οι λειτουργίες κάθε ακροδέκτη δίνονται στην παρακάτω εικόνα:



Εικόνα 14: Ακροδέκτες του LPC2129 [5]

Οι καταχωρητές **PINSEL0**, **PINSEL1** και **PINSEL2** επιλέγουν τη λειτουργία των ακροδεκτών ελέγχοντας τους αντίστοιχους πολυπλέκτες που ρυθμίζουν τη σύνδεση μεταξύ ακροδεκτών και αντίστοιχων περιφερειακών. Η επιλογή της λειτουργίας κάθε αποδέκτη οφείλει να γίνεται πριν τη χρήση του καθώς και του αντίστοιχου περιφερειακού. Οι δυνατές τιμές των καταχωρητών αυτών είναι:

PINSEL0	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.0	GPIO Port 0.0	TxD (UART0)	PWM1	Reserved	00
3:2	P0.1	GPIO Port 0.1	RxD (UART0)	PWM3	EINT0	00
5:4	P0.2	GPIO Port 0.2	SCL (I^2C)	Capture 0.0 (TIMER0)	Reserved	00
7:6	P0.3	GPIO Port 0.3	SDA (I^2C)	Match 0.0 (TIMER0)	EINT1	00
9:8	P0.4	GPIO Port 0.4	SCK (SPI0)	Capture 0.1 (TIMER0)	Reserved	00
11:10	P0.5	GPIO Port 0.5	MISO (SPI0)	Match 0.1 (TIMER0)	Reserved	00
13:12	P0.6	GPIO Port 0.6	MOSI (SPI0)	Capture 0.2 (TIMER0)	Reserved	00
15:14	P0.7	GPIO Port 0.7	SSEL (SPI0)	PWM2	EINT2	00
17:16	P0.8	GPIO Port 0.8	TxD UART1	PWM4	Reserved	00
19:18	P0.9	GPIO Port 0.9	RxD (UART1)	PWM6	EINT3	00
21:20	P0.10	GPIO Port 0.10	RTS (UART1)	Capture 1.0 (TIMER1)	Reserved	00
23:22	P0.11	GPIO Port 0.11	CTS (UART1)	Capture 1.1 (TIMER1)	Reserved	00
25:24	P0.12	GPIO Port 0.12	DSR (UART1)	Match 1.0 (TIMER1)	Reserved	00
27:26	P0.13	GPIO Port 0.13	DTR (UART1)	Match 1.1 (TIMER1)	Reserved	00
29:28	P0.14	GPIO Port 0.14	CD (UART1)	EINT1	Reserved	00
31:30	P0.15	GPIO Port 0.15	RI (UART1)	EINT2	Reserved	00

Πίνακας 9: Καταχωρητής PINSEL0 [5]

PINSEL1	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.16	GPIO Port 0.16	EINT0	Match 0.2 (TIMER0)	Capture 0.2 (TIMER0)	00
3:2	P0.17	GPIO Port 0.17	Capture 1.2 (TIMER1)	SCK (SPI1)	Match 1.2 (TIMER1)	00
5:4	P0.18	GPIO Port 0.18	Capture 1.3 (TIMER1)	MISO (SPI1)	Match 1.3 (TIMER1)	00
7:6	P0.19	GPIO Port 0.19	Match 1.2 (TIMER1)	MOSI (SPI1)	Match 1.3 (TIMER1)	00
9:8	P0.20	GPIO Port 0.20	Match 1.3 (TIMER1)	SSEL (SPI1)	EINT3	00
11:10	P0.21	GPIO Port 0.21	PWM5	Reserved	Capture 1.3 (TIMER1)	00
13:12	P0.22	GPIO Port 0.22	Reserved	Capture 0.0 (TIMER0)	Match 0.0 (TIMER0)	00
15:14	P0.23	GPIO Port 0.23	RD2 (CAN Controller 2)	Reserved	Reserved	00
17:16	P0.24	GPIO Port 0.24	TD2 (CAN Controller 2)	Reserved	Reserved	00
19:18	P0.25	GPIO Port 0.25	RD1 (CAN Controller 1)	Reserved	Reserved	00
21:20	P0.26	Reserved				00
23:22	P0.27	GPIO Port 0.27	AIN0 (A/D Converter)	Capture 0.1 (TIMER0)	Match 0.1 (TIMER0)	01
25:24	P0.28	GPIO Port 0.28	AIN1 (A/D Converter)	Capture 0.2 (TIMER0)	Match 0.2 (TIMER0)	01
27:26	P0.29	GPIO Port 0.29	AIN2 (A/D Converter)	Capture 0.3 (TIMER0)	Match 0.3 (TIMER0)	01
29:28	P0.30	GPIO Port 0.30	AIN3 (A/D Converter)	EINT3	Capture 0.0 (TIMER0)	01
31:30	P0.31	Reserved				00

Πίνακας 10: Καταχωρητής PINSEL1 [5]

PINSEL2	Description	Reset Value
1:0	Reserved.	00
2	When 0, pins P1.36:26 are used as GPIO pins. When 1, P1.31:26 are used as a Debug port.	$\overline{\text{P1.26/RTCK}}$
3	When 0, pins P1.25:16 are used as GPIO pins. When 1, P1.25:16 are used as a Trace port.	$\overline{\text{P1.20/}}$ $\overline{\text{TRACESYNC}}$
4:5	Reserved. **Note:** These bits must not be altered at any time. Changing them may result in an incorrect code execution.	11
6:31	Reserved.	NA

Πίνακας 11: Καταχωρητής PINSEL2 [5]

Είσοδοι/Εξοδοι Γενικής Χρήσης

Ο LPC2129 διαθέτει έως 46 ακροδέκτες που μπορούν να χρησιμοποιηθούν ως γενικής χρήσης ακροδέκτες εισόδου/εξόδου. Καθένας από αυτούς ελέγχεται αυτόνομα και μπορεί να χρησιμοποιηθεί για ανάγνωση ψηφιακών δεδομένων, έλεγχο συσκευών που βρίσκονται εκτός της ψηφίδας του μικροεπεξεργαστή, οδήγηση LEDs ή άλλων συσκευών ένδειξης κ.ά. Οι ακροδέκτες αυτοί είναι χωρισμένοι σε δύο ομάδες που ονομάζονται θύρες (ports). Η θύρα 0 περιλαμβάνει 30 ακροδέκτες και η θύρα 1 άλλους 16. Καθώς οι ίδιοι ακροδέκτες χρησιμοποιούνται και σε άλλα περιφερειακά, η ενεργοποίησή τους ως ακροδέκτες γενικής χρήσης προϋποθέτει ότι οι υπόλοιπες λειτουργίες τους είναι απενεργοποιημένες.

Το σύστημα ελέγχου των Ε/Ε γενικής χρήσης στηρίζεται σε 8 καταχωρητές, 4 για κάθε θύρα. Για ευκολία στον προγραμματισμό έχει υιοθετηθεί η προσέγγιση που προβλέπει ότι ο ορισμός κάθε ακροδέκτη που λειτουργεί ως έξοδος σε μηδενικό ή μονάδα γίνεται από διαφορετικούς καταχωρητές. Κατά την αρχικοποίηση του συστήματος, όλοι οι ακροδέκτες ορίζονται σε είσοδοι. Οι 4 καταχωρητές που υπάρχουν για κάθε θύρα είναι:

Generic Name	Description	Access	Reset Value	PORT0 Address & Name	PORT1 Address & Name	PORT2 Address & Name	PORT3 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode. Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN	0xE0028010 IO1PIN	0xE0028020 IO2PIN	0xE0028030 IO3PIN
IOSET	GPIO Port Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/ Write	0x0000 0000	0xE0028004 IO0SET	0xE0028014 IO1SET	0xE0028024 IO2SET	0xE0028034 IO3SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	Read/ Write	0x0000 0000	0xE0028008 IO0DIR	0xE0028018 IO1DIR	0xE0028028 IO2DIR	0xE0028038 IO3DIR
IOCLR	GPIO Port Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Write Only	0x0000 0000	0xE002800C IO0CLR	0xE002801C IO1CLR	0xE002802C IO2CLR	0xE002803C IO3CLR

*Οι θύρες 2 και 3 δεν είναι διαθέσιμες στον LPC2129

Πίνακας 12: Καταχωρητές Ε/Ε Γενικής Χρήσης [5]

Ο καταχωρητής **IODIR** καθορίζει την κατεύθυνση κάθε ακροδέκτη. Οι ακροδέκτες που αντιστοιχούν σε μονάδες ορίζονται έξοδοι ενώ αυτοί που αντιστοιχούν σε μηδενικά ορίζονται είσοδοι. Ο καταχωρητής **IOPIN** περιέχει τις τιμές των ακροδεκτών που έχουν ορισθεί ως είσοδοι γενικής χρήσης. Παρόλο που ο IOPIN περιγράφεται σαν καταχωρητής μόνο για ανάγνωση, η εγγραφή τιμής σε αυτόν έχει ως αποτέλεσμα τον ορισμό των αντίστοιχων τιμών στους ακροδέκτες που λειτουργούν ως έξοδοι.

Ο καταχωρητής **IOSET** χρησιμοποιείται για να ορίζει σε μονάδα ακροδέκτες που λειτουργούν ως έξοδοι γενικής χρήσης. Η εγγραφή μονάδας σε κάποιο bit του καταχωρητή αυτού προκαλεί τον αντίστοιχο ακροδέκτη εξόδου να αποκτήσει υψηλό δυναμικό, ενώ η εγγραφή μηδενικού αφήνει την κατάστασή του ανεπηρέαστη. Οι ακροδέκτες που είναι ορισμένοι ως είσοδοι ή χρησιμοποιούνται από άλλα περιφερειακά δεν επηρεάζονται. Η ανάγνωση του καταχωρητή επιστρέφει την κατάσταση στην οποία έχουν οριστεί από το πρόγραμμα οι ακροδέκτες εξόδου γενικής χρήσης. Η εγγραφή μονάδας σε bits του καταχωρητή **IOCLR** αντίθετα προκαλεί την εμφάνιση χαμηλού δυναμικού στους αντίστοιχους ακροδέκτες εξόδου, ενώ η εγγραφή μηδενικού δεν αλλάζει την κατάστασή τους.

UART (Universal Asynchronous Receiver/Transmitter)

Γενικά

Το υποσύστημα UART εκτελεί σειριακή μεταφορά δεδομένων μεταξύ δύο συσκευών με ασύγχρονο τρόπο, δηλαδή η αρχή και το τέλος της επικοινωνίας δηλώνεται με bits έναρξης και λήξης, απαίτηση που δεν υπάρχει όταν η επικοινωνία είναι σύγχρονη (στην περίπτωση αυτή βέβαια υπάρχουν άλλα προβλήματα). Κατά σύμβαση, η επικοινωνία ξεκινάει με ένα bit έναρξης, που είναι σε κατάσταση διαφορετική από την κατάσταση αδράνειας της γραμμής μεταφοράς δεδομένων, το οποίο ακολουθούν 5 έως 8 bits δεδομένων με το λιγότερο σημαντικό bit πρώτο, ένα προαιρετικό bit ισοτιμίας και ένα, ενάμισι ή δύο bits τερματισμού, που είναι σε κατάσταση ίδια με την κατάσταση αδράνειας της γραμμής.

Το bit ισοτιμίας χρησιμοποιείται για τον έλεγχο της σωστής μετάδοσης των δεδομένων. Όταν έχει επιλεχθεί η άρτια ισοτιμία, το bit αυτό είναι τέτοιο ώστε το πλήθος των μονάδων που περιλαμβάνονται στα bits των δεδομένων και το bit ισοτιμίας να είναι άρτιο. Αντίστοιχα ισχύουν για την περιττή ισοτιμία, η οποία είναι και προτιμότερη καθώς επιβάλλει τουλάχιστον μία μεταβολή στο επίπεδο τάσης της γραμμής δεδομένων επιτρέποντας έτσι τον καλύτερο συγχρονισμό των δύο συσκευών. Η επιμήκυνση της διάρκειας του bit τερματισμού δίνει χρόνο στο δέκτη να επεξεργαστεί το μήνυμα πριν αρχίσει η μετάδοση ενός επόμενου μηνύματος.

Ο όρος «Universal» στο UART αναφέρεται στη δυνατότητα επιλογής του ρυθμού μεταφοράς δεδομένων ώστε να επιτρέπεται η επικοινωνία συσκευών με διαφορετικές δυνατότητες ταχύτητας επεξεργασίας.

Ο LPC2129 περιλαμβάνει δύο συσκευές UART, από τις οποίες η UART0 έχει όλες τις συνηθισμένες δυνατότητες επικοινωνίας ενώ η UART1 επιτρέπει επιπλέον και την επικοινωνία με modem. Οι συσκευές αυτές λειτουργούν σε στάθμες δυναμικού διαφορετικές από τις στάθμες στη γραμμή μεταφοράς δεδομένων. Η μετάβαση μεταξύ των δύο διαφορετικών αυτών στάθμεων γίνεται με τη χρήση εξωτερικού ολοκληρωμένου (τυπικά χρησιμοποιείται το MAX232) το οποίο επιτρέπει τη σύνδεση των συσκευών σε συστήματα σειριακής επικοινωνίας που ακολουθούν το πρωτόκολλο RS-232. Το πρωτόκολλο αυτό ορίζει τα επίπεδα τάσης που χρησιμοποιούνται στις γραμμές, τα

καλώδια που απαιτούνται και τους τύπους των εξαρτημάτων που χρησιμοποιούνται για τη σύνδεση των συσκευών.

Κατά κανόνα, μια συσκευή UART αποτελείται από:

- Γεννήτρια ρολογιού (που συχνά παράγει περισσότερους από έναν παλμούς σε κάθε bit για καλύτερο συγχρονισμό)
- Καταχωρητές ολίσθησης στην είσοδο και την έξοδο
- Λογική ελέγχου της μετάδοσης και λήψης δεδομένων
- Λογική ελέγχου ανάγνωσης και εγγραφής
- Προαιρετικές μονάδες προσωρινής αποθήκευσης (buffers)
- Προαιρετικό σύστημα FIFO (First-In-First-Out)

Στη σειριακή ασύγχρονη επικοινωνία μπορούν να προκύψουν τέσσερα είδη σφαλμάτων: σφάλμα υπερχείλισης (overrun error) που σημαίνει ότι ο δέκτης δεν είχε προλάβει να επεξεργαστεί το μήνυμα που είχε λάβει όταν ο πομπός έστειλε το επόμενο μήνυμα, σφάλμα πλαισίου (framing error) που συμβαίνει όταν τη στιγμή που ο δέκτης ανέμενε ένα bit τερματισμού η γραμμή δεδομένων δεν ήταν στην αντίστοιχη κατάσταση, σφάλμα ισοτιμίας (parity error) που σημαίνει ότι το bit ισοτιμίας, όταν υπάρχει, δεν συνάδει με την ισοτιμία που έχει ορισθεί και η κατάσταση στάσης (break condition/interrupt) που προκαλείται όταν ο δέκτης διαβάζει από τη γραμμή ένα επίπεδο «στάσης» («break level»), που αντιστοιχεί στο λογικό 0, για περισσότερο από ένα ολόκληρο μήνυμα (bit έναρξης, δεδομένα, bit ισοτιμίας, bit τερματισμού).

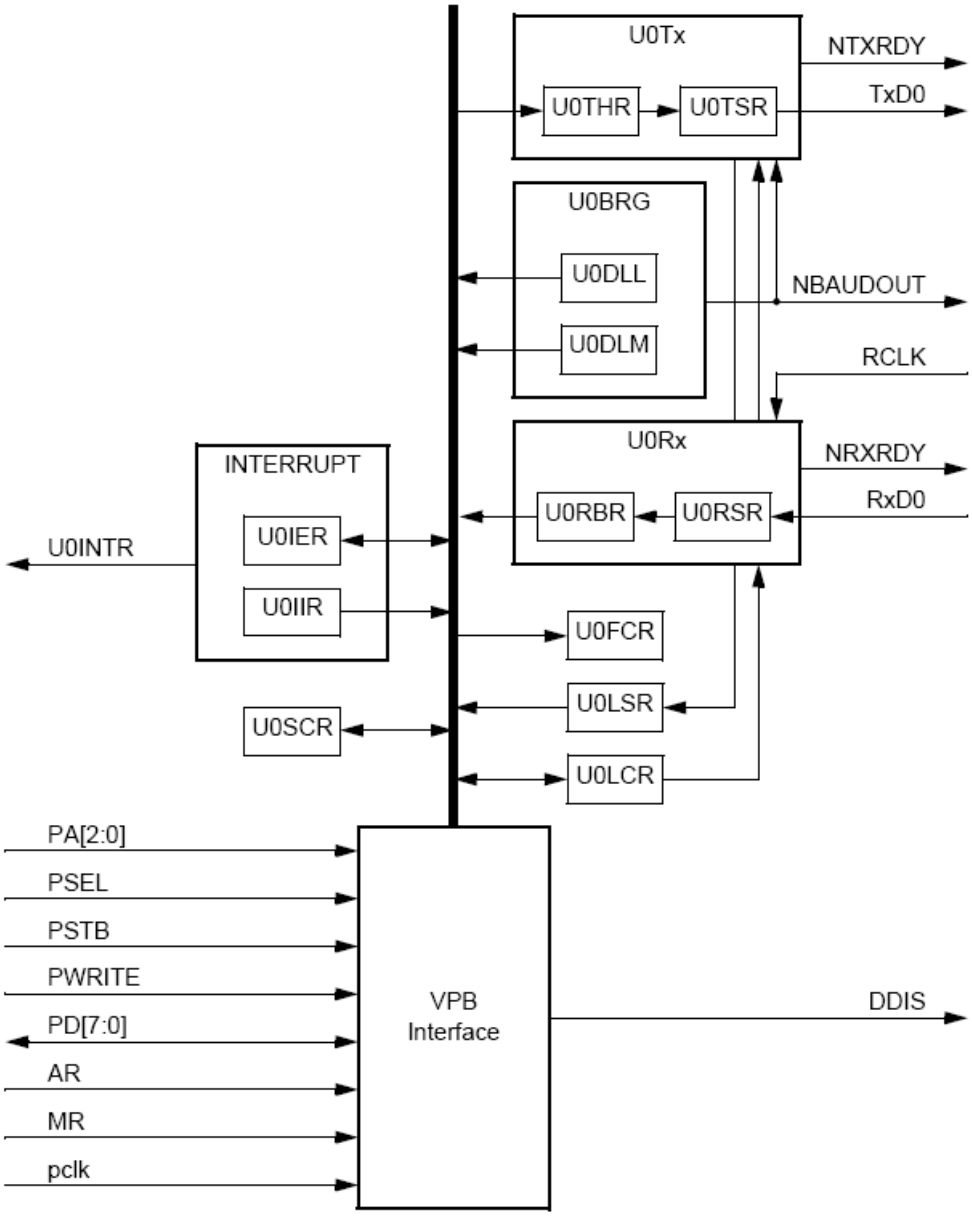
UART 0

Το περιφερειακό UART0 του μικροεπεξεργαστή ελέγχεται από τους παρακάτω καταχωρητές:

Name	Description	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	Access	Reset Value*	Address
U0RBR	Receiver Buffer Register	MSB			READ DATA				LSB	RO	un-defined	0xE000C000 DLAB = 0
U0THR	Transmit Holding Register	MSB			WRITE DATA				LSB	WO	NA	0xE000C000 DLAB = 0
U0IER	Interrupt Enable Register	0	0	0	0	0	Enable Rx Line Status Interrupt	Enable THRE Interrupt	Enable Rx Data Available Interrupt	R/W	0	0xE000C004 DLAB = 0
U0IIR	Interrupt ID Register	FIFOs Enabled		0	0	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE000C008
U0FCR	FIFO Control Register	Rx Trigger		Reserved		-	Tx FIFO Reset	Rx FIFO Reset	FIFO Enable	WO	0	0xE000C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0	0xE000C00C
U0LSR	Line Status Register	Rx FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000C014
U0SCR	Scratch Pad Register	MSB							LSB	R/W	0	0xE000C01C
U0DLL	Divisor Latch LSB	MSB							LSB	R/W	0x01	0xE000C000 DLAB = 1
U0DLM	Divisor Latch MSB	MSB							LSB	R/W	0	0xE000C004 DLAB = 1

Πίνακας 13: Καταχωρητές του UART0 [5]

Η αρχιτεκτονική του συστήματος, το οποίο ακολουθεί την περιγραφή που έγινε νωρίτερα, δίνεται συνοπτικά στο επόμενο σχήμα:



Εικόνα 15: Αρχιτεκτονική του UART0 [5]

Το σύστημα FIFO λήψης (Rx) έχει το νεότερο δεδομένο στον καταχωρητή ολίσθησης U0RSR και το παλιότερο στον καταχωρητή U0RBR. Το σύστημα FIFO αποστολής (Tx) έχει το παλιότερο και άρα πρώτο προς αποστολή δεδομένο στον καταχωρητή ολίσθησης U0TSR και το νεότερο και τελευταίο προς αποστολή δεδομένο στον καταχωρητή U0THR.

Ο καταχωρητής **U0RBR** περιέχει το υψηλότερο byte του συστήματος Rx FIFO, δηλαδή το πιο παλιό από τα δεδομένα που έχουν ληφθεί. Το LSB αποτελεί το bit που είχε

ληφθεί πρώτο στα δεδομένα του μηνύματος αυτού και αν το μήνυμα περιείχε λιγότερα από 8 bits ως δεδομένα, τα MSB τίθενται μηδενικά. Ο καταχωρητής **U0THR** περιέχει το υψηλότερο byte του συστήματος Tx FIFO, δηλαδή το νεότερο από τα δεδόμενα που τοποθετείται στην ουρά αποστολής.

Οι καταχωρητές **U0DLL** και **U0DLM** περιέχουν αντίστοιχα το χαμηλότερο και το υψηλότερο byte του 16-bit διαιρέτη που ελέγχει τη γεννήτρια του ρολογιού. Η συχνότητα του ρολογιού του UART0 υπολογίζεται από τη διαίρεση της εξόδου του VPB (pclk) με το 16-bit διαιρέτη και είναι το 16-πλάσιο της συχνότητας επικοινωνίας της γραμμής, που ονομάζεται baud rate. Η πρόσβαση στους καταχωρητές U0DLL και U0DLM είναι δυνατή μόνο όταν το bit DLAB του καταχωρητή U0CLR είναι μονάδα. Ταυτόχρονα, δεν είναι δυνατή η χρήση των καταχωρητών U0RBR και U0THR και αντίστροφα. Το σχήμα αυτό δεν επιτρέπει την αποστολή και λήψη δεδομένων όταν ρυθμίζεται ο ρυθμός επικοινωνίας της γραμμής και, αντίστροφα, δεν επιτρέπει τη μεταβολή του ρυθμού επικοινωνίας όταν στέλνονται ή διαβάζονται δεδομένα από τη γραμμή.

Ο καταχωρητής **U0IER** χρησιμοποιείται για την ενεργοποίηση των τεσσάρων πηγών διακοπών του UART0 σύμφωνα με τον παρακάτω πίνακα:

U0IER	Function	Description	Reset Value
0	RBR Interrupt Enable	0: Disable the RDA interrupt. 1: Enable the RDA interrupt. U0IER0 enables the Receive Data Available interrupt for UART0. It also controls the Character Receive Time-out interrupt.	0
1	THRE Interrupt Enable	0: Disable the THRE interrupt. 1: Enable the THRE interrupt. U0IER1 enables the THRE interrupt for UART0. The status of this interrupt can be read from U0LSR5.	0
2	Rx Line Status Interrupt Enable	0: Disable the Rx line status interrupts. 1: Enable the Rx line status interrupts. U0IER2 enables the UART0 Rx line status interrupts. The status of this interrupt can be read from U0LSR[4:1].	0
7:3	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Πίνακας 14: Περιεχόμενα του U0IER [5]

Ο καταχωρητής **U0IIR** περιέχει πληροφορίες για την πηγή και την προτεραιότητα μιας εκκρεμούς διακοπής σύμφωνα με τον πίνακα:

U0IIR	Function	Description	Reset Value
0	Interrupt Pending	0: At least one interrupt is pending. 1: No pending interrupts. Note that U0IIR0 is active low. The pending interrupt can be determined by evaluating U0IER3:1.	1
3:1	Interrupt Identification	011: 1. Receive Line Status (RLS) 010: 2a.Receive Data Available (RDA) 110: 2b.Character Time-out Indicator (CTI) 001: 3. THRE Interrupt. U0IER3 identifies an interrupt corresponding to the UART0 Rx FIFO. All other combinations of U0IER3:1 not listed above are reserved (000,100,101,111).	0
5:4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	FIFO Enable	These bits are equivalent to U0FCR0.	0

Πίνακας 15: Αναγνώριση πηγής διακοπής με τον U0IIR [5]

Οι τιμές των bits 3:1 του καταχωρητή αυτού χρησιμοποιούνται από τη ρουτίνα εξυπηρέτησης της διακοπής για την αναγνώριση της αιτίας που την προκάλεσε. Πριν την έξοδο από τη ρουτίνα αυτή πρέπει να γίνει μια ανάγνωση του καταχωρητή προκειμένου το bit 0 να γίνει μονάδα και η διακοπή να θεωρηθεί ότι εξυπηρετήθηκε.

Οι αιτίες των διακοπών είναι:

- RLS: έχει την υψηλότερη προτεραιότητα και ενεργοποιείται όταν ένα τουλάχιστον από τα τέσσερα πιθανά σφάλματα επικοινωνίας έχει εμφανιστεί (Overrun Error-OE, Framing Error-FE, Parity Error-PE, Break Interrupt-BI).
- RDA: βρίσκεται στο δεύτερο επίπεδο προτεραιότητας και είναι ενεργή όταν το Rx FIFO φτάνει στο όριο που τίθεται από τα bits 7:6 του U0FCR (βλ. παρακάτω), δηλαδή όταν το FIFO λήψης είναι γεμάτο. Σε αυτήν την περίπτωση, ο μικροεπεξεργαστής μπορεί να διαβάσει ένα μπλοκ δεδομένων μεγέθους ίσου με το όριο που έχει οριστεί και αποτελεί το βάθος του Rx FIFO.
- CTI: βρίσκεται επίσης στο δεύτερο επίπεδο προτεραιότητας και ενεργοποιείται όταν το Rx FIFO περιέχει τουλάχιστον ένα χαρακτήρα αλλά δεν υπάρχει δραστηριότητα λήψης για περίπου 4 χρόνους μηνυμάτων. Η διακοπή αυτή χρησιμοποιείται όταν η ανάγνωση του Rx FIFO γίνεται κατά μπλοκ (μεγέθους όσο το βάθος του Rx FIFO) αλλά ο αποστολέας δεν στέλνει αριθμό μηνυμάτων που είναι ακέραιο πολλαπλάσιο του μπλοκ. Έτσι, είναι πιθανό η αποστολή να διακοπεί και

τα τελευταία μηνύματα του αποστολέα να μην έχουν γεμίσει το Rx FIFO. Στην περίπτωση αυτή, η διακοπή CTI θα επιτρέψει στο μικροεπεξεργαστή να διαβάσει και το τελευταίο, μειωμένο μπλοκ μηνυμάτων.

- THRE: βρίσκεται στο τρίτο και χαμηλότερο επίπεδο προτεραιότητας και προκαλείται όταν ο Tx FIFO είναι άδειος και εφόσον έχει περάσει ένα χρονικό διάστημα που επιτρέπει στην ΚΜΕ να γράψει τον επόμενο προς αποστολή χαρακτήρα στον U0THR.

Συνοπτικά, η αντιμετώπιση των διακοπών αυτών δίνεται στον παρακάτω πίνακα.

U0IIR[3:0]	Priority	Interrupt Type	Interrupt Source	Interrupt Reset
0001	-	none	none	-
0110	Highest	Rx Line Status / Error	OE or PE or FE or BI	U0LSR Read
0100	Second	Rx Data Available	Rx data available or trigger level reached in FIFO (U0FCR0=1)	U0RBR Read or UART0 FIFO drops below trigger level
1100	Second	Character Time-out Indication	Minimum of one character in the Rx FIFO and no character input or removed during a time period depending on how many characters are in FIFO and what the trigger level is set at (3.5 to 4.5 character times). The exact time will be: [(word length) X 7 - 2] X 8 + {(trigger level - number of characters) X 8 + 1] RCLKs	U0 RBR Read
0010	Third	THRE	THRE	U0IIR Read (if source of interrupt) or THR write
note: values "0000", “0011”, “0101”, “0111”, “1000”, “1001”, “1010”, “1011”,”1101”,”1110”,”1111” are reserved.				

Πίνακας 16: Αντιμετώπιση διακοπών του UART0 [5]

Ο καταχωρητής **U0FCR** ελέγχει τη λειτουργία των δύο FIFO του UART0 και ο καταχωρητής **U0LCR** καθορίζει τη μορφή των μηνυμάτων που στέλνονται ή λαμβάνονται. Στους δύο πίνακες φαίνεται η λειτουργία τους.

U0FCR	Function	Description	Reset Value
0	FIFO Enable	Active high enable for both UART0 Rx and Tx FIFOs and U0FCR7:1 access. This bit must be set for proper UART0 opearation. Any transition on this bit will automatically clear the UART0 FIFOs.	0
1	Rx FIFO Reset	Writing a logic 1 to U0FCR1 will clear all bytes in UART0 Rx FIFO and reset the pointer logic. This bit is self-clearing.	0
2	Tx FIFO Reset	Writing a logic 1 to U0FCR2 will clear all bytes in UART0 Tx FIFO and reset the pointer logic. This bit is self-clearing.	0
5:3	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	Rx Trigger Level Select	00: trigger level 0 (default=1 character or 0x01h) 01: trigger level 1 (default=4 characters or 0x04h) 10: trigger level 2 (default=8 characters or 0x08h) 11: trigger level 3 (default=14 characters or 0x0eh) These two bits determine how many receiver UART0 FIFO characters must be written before an interrupt is activated. The four trigger levels are defined by the user at compilation allowing the user to tune the trigger levels to the FIFO depths chosen.	0

Πίνακας 17: Ρύθμιση των FIFO με τον U0FCR [5]

U0LCR	Function	Description	Reset Value
1:0	Word Length Select	00: 5 bit character length 01: 6 bit character length 10: 7 bit character length 11: 8 bit character length	0
2	Stop Bit Select	0: 1 stop bit 1: 2 stop bits (1.5 if U0LCR[1:0]=00)	0
3	Parity Enable	0: Disable parity generation and checking 1: Enable parity generation and checking	0
5:4	Parity Select	00: Odd parity 01: Even parity 10: Forced "1" stick parity 11: Forced "0" stick parity	0
6	Break Control	0: Disable break transmission 1: Enable break transmission. Output pin UART0 TxD is forced to logic 0 when U0LCR6 is active high.	0
7	Divisor Latch Access Bit	0: Disable access to Divisor Latches 1: Enable access to Divisor Latches	0

Πίνακας 18: Ρύθμιση της γραμμής με τον U0LCR [5]

Ο καταχωρητής **U0LSR** είναι μόνο για ανάγνωση και περιέχει την κατάσταση της γραμμής και των υποσυστημάτων αποστολής και λήψης μηνυμάτων.

U0LSR	Function	Description	Reset Value
0	Receiver Data Ready (RDR)	0: U0RBR is empty 1: U0RBR contains valid data U0LSR0 is set when the U0RBR holds an unread character and is cleared when the UART0 RBR FIFO is empty.	0
1	Overrun Error (OE)	0: Overrun error status is inactive. 1: Overrun error status is active. The overrun error condition is set as soon as it occurs. An U0LSR read clears U0LSR1. U0LSR1 is set when UART0 RSR has a new character assembled and the UART0 RBR FIFO is full. In this case, the UART0 RBR FIFO will not be overwritten and the character in the UART0 RSR will be lost.	0
2	Parity Error (PE)	0: Parity error status is inactive. 1: Parity error status is active. When the parity bit of a received character is in the wrong state, a parity error occurs. An U0LSR read clears U0LSR2. Time of parity error detection is dependent on U0FCR0. A parity error is associated with the character being read from the UART0 RBR FIFO.	0
3	Framing Error (FE)	0: Framing error status is inactive. 1: Framing error status is active. When the stop bit of a received character is a logic 0, a framing error occurs. An U0LSR read clears U0LSR3. The time of the framing error detection is dependent on U0FCR0. A framing error is associated with the character being read from the UART0 RBR FIFO. Upon detection of a framing error, the Rx will attempt to resynchronize to the data and assume that the bad stop bit is actually an early start bit. However, it cannot be assumed that the next received byte will be correct even if there is no Framing Error.	0
4	Break Interrupt (BI)	0: Break interrupt status is inactive. 1: Break interrupt status is active. When RxD0 is held in the spacing state (all 0's) for one full character transmission (start, data, parity, stop), a break interrupt occurs. Once the break condition has been detected, the receiver goes idle until RxD0 goes to marking state (all 1's). An U0LSR read clears this status bit. The time of break detection is dependent on U0FCR0. The break interrupt is associated with the character being read from the UART0 RBR FIFO.	0
5	Transmitter Holding Register Empty (THRE)	0: U0THR contains valid data. 1: U0THR is empty. THRE is set immediately upon detection of an empty UART0 THR and is cleared on a U0THR write.	1
6	Transmitter Empty (TEMT)	0: U0THR and/or the U0TSR contains valid data. 1: U0THR and the U0TSR are empty. TEMT is set when both U0THR and U0TSR are empty; TEMT is cleared when either the U0TSR or the U0THR contain valid data.	1
7	Error in Rx FIFO (RXFE)	0: U0RBR contains no UART0 Rx errors or U0FCR0=0. 1: UART0 RBR contains at least one UART0 Rx error. U0LSR7 is set when a character with a Rx error such as framing error, parity error or break interrupt, is loaded into the U0RBR. This bit is cleared when the U0LSR register is read and there are no subsequent errors in the UART0 FIFO.	0

Πίνακας 19: Καταχωρητής U0LSR [5]

Τέλος, υπάρχει διαθέσιμος και ένας καταχωρητής των 8 bits, **U0SCR**, ο οποίος μπορεί να χρησιμοποιηθεί από τον προγραμματιστή σαν πρόχειρο, χωρίς να έχει καμμία αλληλεπίδραση με κάποιον άλλον καταχωρητή ή με κάποιο υποσύστημα του μικροεπεξεργαστή (εκτός από τη μνήμη, φυσικά!).

Ο LPC2129 διαθέτει και δεύτερη συσκευή UART, την UART1, η οποία περιλαμβάνει όλες τις λειτουργίες της UART0 με επιπλέον τη δυνατότητα διασύνδεσης modem. Οι κανονικές λειτουργίες υλοποιούνται μέσω των ίδιων καταχωρητών όπως της

UART0, με τη διαφορά ότι η ονομασία τους ξεκινάει με U1. Η λειτουργία που σχετίζεται με modem χρησιμοποιεί κάποιους ειδικούς καταχωρητές, η χρήση των οποίων δεν θα αναλυθεί καθώς δεν χρησιμοποιούνται στην εφαρμογή που αναπτύσσεται.

Ελεγκτές CAN

Γενικά

Το CAN (Controller Area Network) είναι ένα σειριακό, διαφορικό πρωτόκολλο επικοινωνίας μεταξύ πολλαπλών συσκευών που υποστηρίζει αποτελεσματικά απομακρυσμένο έλεγχο σε πραγματικό χρόνο. Αναπτύχθηκε μέσα στη δεκαετία του 1980 και σχεδιάστηκε ειδικά ώστε να είναι ανθεκτικό σε περιβάλλοντα με έντονο ηλεκτρομαγνητικό θόρυβο και να προσφέρει μεγάλη ασφάλεια στη μεταφορά δεδομένων. Τα μηνύματα που μεταδίδονται είναι σχετικά μικρά, με μέγιστο πλήθος 8 bytes δεδομένων, αλλά προστατεύονται από 15 bit CRC το οποίο εξασφαλίζει ότι 5 συνεχόμενα εσφαλμένα bits θα γίνουν αντιληπτά από κάθε κόμβο CAN που είναι συνδεδεμένος στο δίαυλο. Το CAN χρησιμοποιήθηκε αρχικά σε οχήματα αλλά η χρήση του πλέον διευρύνεται σε όλες τις εφαρμογές αυτομάτου ελέγχου είτε για επικοινωνία σε μεγάλη ταχύτητα είτε για μείωση της καλωδίωσης, καθώς συνήθως απαιτεί μόνο δύο αγωγούς για τη μεταφορά των δεδομένων. Εναλλακτικά, ο δίαυλος CAN μπορεί να χρησιμοποιεί μόνο ένα καλώδιο (με μειωμένη ταχύτητα και αξιοπιστία), οπτικές ίνες, ασύρματη μετάδοση και γίνονται προσπάθειες για τη μεταφορά σημάτων CAN διαμέσου γραμμών ισχύος.

Η ταχύτητα μεταφοράς δεδομένων μπορεί να φτάσει το 1Mbit/s για μήκη δικτύου έως 40 μέτρα αλλά η ελάττωση της ταχύτητας επιτρέπει μεγαλύτερες αποστάσεις.

Το πρωτόκολλο έχει τυποποιηθεί με το πρότυπο ISO 11898-1 του 1993, στο οποίο περιγράφεται κυρίως ο τρόπος μετάδοσης δεδομένων (data link layer) και η φυσική διασύνδεση του διαύλου (physical layer). Σε ανώτερο επίπεδο υπάρχουν διάφορα πρωτόκολλα, με το CANOpen να είναι το πλέον διαδεδομένο.

Στα πλαίσια της εργασίας αυτής θα γίνει μια σύντομη αναφορά στα χαρακτηριστικά του πρωτοκόλλου, τα οποία υλοποιούνται με τους καταχωρητές των ελεγκτών CAN που είναι ενσωματωμένοι στον LPC2129. Περισσότερες πληροφορίες είναι διαθέσιμες στη βιβλιογραφία που παρατίθεται στο τέλος καθώς και στο διαδίκτυο.

Τα δυαδικά δεδομένα που μεταφέρονται σε ένα δίαυλο CAN, ο οποίος αποτελείται από δύο αγωγούς, είναι είτε τύπου «επικρατών» («dominant») είτε τύπου «υποχωρών» («recessive»). Τα dominant bits αντιστοιχούν στο λογικό 0 και τα recessive στο λογικό 1. Η μετάδοση δεν απαιτεί μετάδοση σήματος ρολογιού. Τα μηνύματα που μεταδίδονται διαμέσου ενός διαύλου CAN ονομάζονται «πλαίσια» («frames») και αποτελούνται από ένα αναγνωριστικό πεδίο, ένα πεδίο δεδομένων και κάποια πεδία ελέγχου.

Ένα από τα ιδιαίτερα χαρακτηριστικά του πρωτοκόλλου είναι ότι κάθε μήνυμα έχει το δικό του αναγνωριστικό, το οποίο χρησιμοποιείται ταυτόχρονα για τη διαιτησία του διαύλου (δηλαδή την απόφαση για το ποιο μήνυμα θα σταλεί όταν περισσότεροι του ένος κόμβοι προσπαθούν να στείλουν ένα μήνυμα) και για την απόφαση των δεκτών για το αν θα το λάβουν υπ'όψην τους ή θα το αγνοήσουν. Η αναγνώριση της διαιτησίας γίνεται με την παρατήρηση του διαύλου από κάθε κόμβο που αποστέλει ένα μήνυμα. Όταν στο δίαυλο μεταδίδονται ταυτόχρονα ένα «dominant» και ένα «recessive» bit, το «dominant» bit είναι αυτό που θα επικρατήσει. Ο κόμβος που απέστειλε το «recessive» bit θα αναγνωρίσει ότι η προσπάθειά του απέτυχε και θα περιμένει να τελειώσει η αποστολή του άλλου μηνύματος λειτουργώντας ως δέκτης πριν προσπαθήσει να ξαναστείλει το μήνυμά του.

Κάθε κόμβος μπορεί να αποστέλει είτε ένα μήνυμα δεδομένων (data frame) είτε ένα μήνυμα αίτησης δεδομένων (remote frame). Υπάρχουν δύο δομές πλαισίων που διαφοροποιούνται στο μέγεθος του αναγνωριστικού πεδίου. Η βασική δομή (CAN 2.0 A) έχει αναγνωριστικό μήκους 11 bits ενώ η εκτεταμένη δομή (CAN 2.0 B) έχει αναγνωριστικό μήκους 29 bits. Η βασική δομή ενός πλαισίου CAN αποτελείται από:

- 1 bit που δηλώνει την έναρξη του πλαισίου (Start-Of-Frame)
- 11 bits αναγνωριστικό πεδίο (Identifier)
- 1 bit αίτησης δεδομένων (Remote Transmission Request) (είναι «dominant» για αποστολή και «recessive» για αίτηση αποστολής)
- 1 bit επέκτασης αναγνωριστικού (Identifier Extension Bit) (είναι «dominant» για τη βασική δομή)
- 1 αχρησιμοποίητο bit (reserved) (είναι «dominant»)
- 4 bits που δηλώνουν το πλήθος των δεδομένων που αποστέλλονται ή αιτούνται (Data Length Code) (είναι μεταξύ 0 και 8 bytes)
- 0 έως 8 bytes δεδομένων (Data)

- 15 bits ελέγχου κυκλικής επαναφοράς δεδομένων (Cyclic Redundancy Check)
- 1 bit διαχωριστικού CRC (CRC delimiter) (είναι «recessive»)
- 1 bit βεβαίωσης λήψης (ACK slot) (ο αποστολέας στέλνει «recessive» και οποιοσδήποτε δέκτης το κάνει «dominant»)
- 1 bit διαχωριστικού βεβαίωσης λήψης (ACK delimiter) (είναι «recessive»)
- 7 bits που δηλώνουν το τέλος του πλαισίου (End-Of-Frame) (είναι όλα «recessive»)

Αντίστοιχα, η εκτεταμένη δομή ενός πλαισίου CAN αποτελείται από:

- 1 bit που δηλώνει την έναρξη του πλαισίου (Start-Of-Frame)
- 11 bits πρώτο μέρος του αναγνωριστικού πεδίου (Identifier A)
- 1 bit αίτησης δεδομένων (Remote Transmission Request) (είναι «dominant» για αποστολή και «recessive» για αίτηση αποστολής)
- 1 bit επέκτασης αναγνωριστικού (Identifier Extension Bit) (είναι «recessive» για την εκτεταμένη δομή)
- 18 bits δεύτερο μέρος του αναγνωριστικού πεδίου (Identifier B)
- 2 αχρησιμοποίητα bits (reserved) (είναι «dominant»)
- 4 bits που δηλώνουν το πλήθος των δεδομένων που αποστέλλονται ή αιτούνται (Data Length Code) (είναι μεταξύ 0 και 8 bytes)
- 0 έως 8 bytes δεδομένων (Data)
- 15 bits ελέγχου κυκλικής επαναφοράς δεδομένων (Cyclic Redundancy Check)
- 1 bit διαχωριστικού CRC (CRC delimiter) (είναι «recessive»)
- 1 bit βεβαίωσης λήψης (ACK slot) (ο αποστολέας στέλνει «recessive» και οποιοσδήποτε δέκτης το κάνει «dominant»)
- 1 bit διαχωριστικού βεβαίωσης λήψης (ACK delimiter) (είναι «recessive»)
- 7 bits που δηλώνουν το τέλος του πλαισίου (End-Of-Frame) (είναι όλα «recessive»)

Οι κόμβοι που υποστηρίζουν την εκτεταμένη δομή υποχρεωτικά υποστηρίζουν και τη βασική δομή, ενώ όταν υπάρχει πρόβλημα διαιτησίας μεταξύ μηνυμάτων διαφορετικής δομής (καθώς επιτρέπεται να χρησιμοποιούνται και οι δύο δομές στον ίδιο δίαυλο), τα μηνύματα που έχουν τη βασική δομή επικρατούν. Οι κόμβοι που δεν υποστηρίζουν την

εκτεταμένη δομή είτε θεωρούν εσφαλμένο ένα εκτεταμένο πλαίσιο είτε το αναγνωρίζουν αλλά το αγνοούν.

Η χρήση πλαισίων εκτεταμένης δομής έχει το προφανές πλεονέκτημα περισσότερων επιλογών στο αναγνωριστικό πεδίο αλλά και τα εξής μειονεκτήματα:

- Ο χρόνος αποστολής του μηνύματος είναι μεγαλύτερος
- Απαιτείται μεγαλύτερο εύρος ζώνης (bandwidth)
- Η αναγνώριση σφαλμάτων είναι λιγότερο αποτελεσματική καθώς το 15 bit CRC είναι βέλτιστο για τα πλαίσια βασικής δομής

Η αναγνώριση σφαλμάτων γίνεται στο επίπεδο του bit και στο επίπεδο του μηνύματος. Στο επίπεδο του bit χρησιμοποιούνται δύο μηχανισμοί:

- Παρακολούθηση διαύλου: κάθε κόμβος που στέλνει ένα μήνυμα ταυτόχρονα παρακολουθεί την κατάσταση του διαύλου ώστε, εκτός από τον έλεγχο της διαιτησίας, να αναγνωρίζει αν τα bits που στέλνει μεταφέρονται σωστά στο δίαυλο.
- «Γέμιση» bits («bit stuffing»): το πρωτόκολλο CAN χρησιμοποιεί κωδικοποίηση NRZ (Non Return to Zero) που σημαίνει ότι όποτε ο κόμβος στέλνει 5 συνεχόμενα bits ίδιας τιμής στέλνει και ένα συμπληρωματικής τιμής, το οποίο και αγνοείται από τους δέκτες.

Στο επίπεδο του μηνύματος χρησιμοποιούνται τρεις μηχανισμοί:

- Έλεγχος κυκλικής επαναφοράς (CRC): Κάθε μήνυμα που στέλνεται συνοδεύεται από μία ακολουθία 15 bit που σχετίζεται με συγκεκριμένο τρόπο με τα δεδομένα που περιέχονται στο μήνυμα. Κάθε δέκτης υπολογίζει ξανά την ακολουθία ελέγχου με τον ίδιο τρόπο και τη συγκρίνει με την ακολουθία που συνόδευε το μήνυμα ώστε αν υπάρχει διαφορά να θεωρείται λανθασμένη η αποστολή.
- Έλεγχος πλαισίου: Κάθε κόμβος που δέχεται ένα μήνυμα ελέγχει τη δομή του και εντοπίζει σφάλμα αν η δομή αυτή διαφέρει από την καθορισμένη.
- Σφάλμα ACK: Οι κόμβοι που δέχονται ένα μήνυμα βεβαιώνουν τη λήψη του αλλάζοντας την τιμή του αντίστοιχου bit. Αν ο πομπός δεν εντοπίσει τέτοια αλλαγή, θεωρεί ότι το μήνυμα δεν εστάλει σωστά και αναγνωρίζει το αντίστοιχο σφάλμα ACK.

Η αναγνώριση οποιουδήποτε σφάλματος με έναν από τους παραπάνω μηχανισμούς προκαλεί την αποστολή ενός μηνύματος σφάλματος από τον κόμβο που αναγνώρισε το

σφάλμα. Το μήνυμα αυτό εμποδίζει την ανάγνωση του εσφαλμένου μηνύματος από τους υπόλοιπους κόμβους και στη συνέχεια ο αποστολέας δοκιμάζει αυτόματα να στείλει ξανά το μήνυμά του.

Εδώ πρέπει να σημειωθεί ένα ιδιαίτερο χαρακτηριστικό που προκύπτει από τον τρόπο λειτουργίας του πρωτοκόλλου και μπορεί να προκαλέσει προβλήματα στη λειτουργία του διαύλου. **Σε ένα δίκτυο κόμβων CAN, κάθε αναγνωριστικό (identifier) οφείλει να παράγεται μόνο από ένα συγκεκριμένο κόμβο**. Ο περιορισμός αυτός προκύπτει από τον τρόπο με τον οποίο γίνεται η διαιτησία του διαύλου σε συνδυασμό με την αναγνώριση σφαλμάτων στο επίπεδο του bit με την παρακολούθηση του διαύλου από τον αποστολέα. Συγκεκριμένα, αν δύο διαφορετικοί κόμβοι αποφασίσουν να στείλουν την ίδια στιγμή δύο διαφορετικά μηνύματα με ίδιο αναγνωριστικό, τότε θα κερδίσουν και οι δύο την προτεραιότητα, οπότε θα αρχίσουν και οι δύο να αποστέλλουν το μήνυμά τους. Όταν κάποιο bit δεδομένων διαφέρει μεταξύ των δύο μηνυμάτων, τότε θα αναγνωριστεί σφάλμα παρακολούθησης διαύλου από έναν από τους δύο κόμβους με αποτέλεσμα ο κόμβος αυτός να αποστείλει μήνυμα σφάλματος εμποδίζοντας και το άλλο μήνυμα να ολοκληρωθεί. Αμέσως μόλις τελειώσει το μήνυμα σφάλματος, οι δύο αποστολείς θα προσπαθήσουν ξανά να στείλουν ταυτόχρονα τα ίδια μηνύματα που έστειλαν προηγουμένως, αφού αυτά δεν ολοκληρώθηκαν. Αυτό θα οδηγήσει σε νέο σφάλμα και συνολικά ο δίαυλος θα βρεθεί σε αέναο βρόχο.

Καταχωρητές των ελεγκτών CAN στον LPC2129

Οι δύο ελεγκτές CAN που είναι ενσωματωμένοι στον LPC2129 έχουν κάποιους καταχωρητές ελέγχου των λειτουργιών τους καθώς και ένα κοινό μπλοκ καταχωρητών που καθορίζουν το καθολικό φιλτράρισμα των μηνυμάτων και μερικές ακόμα κεντρικές λειτουργίες. Το κοινό αυτό μπλοκ δίνεται στον παρακάτω πίνακα.

Name	Description	Access	Reset Value	Address
AFMR	Acceptance Filter Register	R/W	1	0xE003 C000
SFF_sa	Standard Frame Individual Start Address Register	R/W	0	0xE003 C004
SFF_GRP_sa	Standard Frame Group Start Address Register	R/W	0	0xE003 C008
EFF_sa	Extended Frame Start Address Register	R/W	0	0xE003 C00C
EFF_GRP_sa	Extended Frame Group Start Address Register	R/W	0	0xE003 C010
ENDofTable	End of AF Tables register	R/W	0	0xE003 C014
LUTerrAd	LUT Error Address register	RO	0	0xE003 C018
CANTxSR	CAN Central Transmit Status Register	RO	0x003F 3F00	0xE004 0000
CANRxSR	CAN Central Receive Status Register	RO	0	0xE004 0004
CANMSR	CAN Central Miscellaneous Register	RO	0	0xE004 0008

Πίνακας 20: Κεντρικοί καταχωρητές CAN [5]

Ο έλεγχος κάθε καναλιού CAN ξεχωριστά γίνεται μέσα από μια σειρά καταχωρητών που ρυθμίζουν τη λειτουργία και καθορίζουν την κατάσταση των αντίστοιχων ελεγκτών και υπάρχει από ένα πανομοιότυπο σύνολο αυτών των καταχωρητών για κάθε ελεγκτή. Αρχικά θα γίνει μια αναφορά σε κάθε έναν από τους καταχωρητές ελέγχου και στη συνέχεια θα περιγραφεί η διαδικασία που ακολουθείται για την αποστολή και λήψη μηνυμάτων και το ρόλο συγκεκριμένων bits των καταχωρητών στη διαδικασία αυτή.

Ο καταχωρητής **CANMOD** καθορίζει την κατάσταση του ελεγκτή CAN. Όταν το LSB του, που ονομάζεται RM (Reset Mode), είναι μονάδα, κάποια από τα υπόλοιπα bits του καταχωρητή αυτού καθώς και άλλων καταχωρητών του ελεγκτή επηρεάζονται. Η στήλη με τίτλο «RM Set» στους παρακάτω πίνακες δίνει την τιμή των αντίστοιχων bits όταν το RM είναι μονάδα. Επιπλέον, όταν το RM είναι μονάδα, οι καταχωρητές που σε κατάσταση κανονικής λειτουργίας είναι μόνο για ανάγνωση, μπορούν να εγγραφούν. Η σημασία κάθε bit αναλύεται στον παρακάτω πίνακα.

CANMOD	Name	Function	Reset Value	RM Set
0	RM	0: the CAN Controller operates, and certain registers can not be written. 1: Reset Mode -- CAN operation is disabled, and writable registers can be written.	1	1
1	LOM	0: the CAN controller acknowledges a successfully-received message on its CAN. 1: Listen Only Mode -- the controller gives no acknowledgment on CAN, even if a message is successfully received. Messages cannot be sent, and the controller operates in "error passive" mode. This mode is intended for software bit rate detection and "hot plugging".	0	X
2	STM	0: a transmitted message must be acknowledged to be considered successful. 1: Self Test Mode -- the controller will consider a Tx message successful if there is no acknowledgment. Use this state in conjunction with the SRR bit in CANCMR.	0	X
3	TPM	0: the priority of the 3 Transmit Buffers depends on their CAN IDs. 1: the priority of the 3 Transmit Buffers depends on their Tx Priority fields.	0	X
4	SM	0: normal operation 1: Sleep Mode -- the CAN controller sleeps if it is not requesting an interrupt, and there is no bus activity. See the Sleep Mode description on page 203.	0	0
5	RPM	0: RX and TX pins are Low for a dominant bit. 1: Reverse Polarity Mode -- RX pins are High for a dominant bit.	0	0
7	TM	0: normal operation 1: Test Mode. The state of the RX pin is clocked onto the TX pin.	0	0

Πίνακας 21: Καταχωρητής CANMOD [5]

Στον παραπάνω καταχωρητή, τα bits LOM και STM μπορούν να εγγραφούν μόνο όταν ο ελεγκτής βρίσκεται σε κατάσταση επανεκκίνησης (Reset Mode=1).

Ο καταχωρητής **CANCMR** είναι μόνο για εγγραφή και κάθε εγγραφή σε κάποιο από τα 8 χαμηλότερα bits του προκαλεί μια ενέργεια σύμφωνα με τον πίνακα που ακολουθεί. Τα υπόλοιπα bits πρέπει να εγγράφονται μηδενικά.

CANCMR	Name	Function
0	TR	1: Transmission Request -- the message, previously written to the CANTFI, CANTID, and optionally the CANTDA and CANTDB registers, is queued for transmission.
1	AT	1: Abort Transmission -- if not already in progress, a pending Transmission Request is cancelled. If this bit and TR are set in the same write operation, frame transmission is attempted once, and no retransmission is attempted if an error is flagged nor if arbitration is lost.
2	RRB	1: Release Receive Buffer -- the information in the CANRFS, CANRID, and if applicable the CANRDA and CANRDB registers is released, and becomes eligible for replacement by the next received frame. If the next received frame is not available, writing this command clears the RBS bit in CANSR.
3	CDO	1: Clear Data Overrun -- The Data Overrun bit in CANSR is cleared.
4	SRR	1: Self Reception Request -- the message, previously written to the CANTFS, CANTID, and optionally the CANTDA and CANTDB registers, is queued for transmission. This differs from the TR bit above in that the receiver is not disabled during the transmission, so that it receives the message if its Identifier is recognized by the Acceptance Filter.
5	STB1	1: Select Tx Buffer 1 for transmission
6	STB2	1: Select Tx Buffer 2 for transmission
7	STB3	1: Select Tx Buffer 3 for transmission

Πίνακας 22: Καταχωρητής CANCMR [5]

Ο καταχωρητής περιέχει πληροφορίες σχετικά με την κατάσταση του ελεγκτή CAN και είναι γενικά μόνο για ανάγνωση, εκτός από τους μετρητές σφαλμάτων οι οποίοι μπορούν να εγγραφούν όταν το RM bit του καταχωρητή CANMOD είναι μονάδα, δηλαδή όταν ο ελεγκτής CAN είναι σε κατάσταση επανεκκίνησης (Reset Mode). Οι λειτουργίες των bits του καταχωρητή δίνονται στον παρακάτω πίνακα, ενώ τα bits που δεν χρησιμοποιούνται διαβάζονται και πρέπει να γράφονται ως μηδενικά.

CANGSR	Name	Function	Reset Value	RM Set
0	RBS	1: Receive Buffer Status -- a received message is available in the CANRFS, CANRID, and if applicable the CANRDA and CANRDB registers. This bit is cleared by the Release Receive Buffer command in CANCMR, if no subsequent received message is available.	0	0
1	DOS	1: Data Overrun Status -- a message was lost because the preceding message to this CAN controller was not read and released quickly enough. 0: No data overrun has occurred since the last Clear Data Overrun command was written to CANCMR (or since Reset).	0	0
2	TBS	1: Transmit Buffer Status -- no transmit message is pending for this CAN controller (in any of the 3 Tx buffers), and software may write to any of the CANTFI, CANTID, CANTDA, and CANTDB registers. 0: as least one previously-queued message for this CAN controller has not yet been sent, and therefore software should not write to the CANTFI, CANTID, CANTDA, nor CANTDB registers of that (those) Tx buffer(s).	1	X
3	TCS	1: Transmit Complete Status -- all requested transmission(s) has (have) been successfully completed. 0: at least one requested transmission has not been successfully completed.	1	0
4	RS	1: Receive Status: the CAN controller is receiving a message.	0	0
5	TS	1: Transmit Status: The CAN controller is sending a message	0	0
6	ES	1: Error Status: one or both of the Transmit and Receive Error Counters has reached the limit set in the Error Warning Limit register.	0	0
7	BS	1: Bus Status: the CAN controller is currently prohibited from bus activity because the Transmit Error Counter reached its limiting value of 255.	0	0
23:16	RXERR	The current value of the Rx Error Counter.	0	X
31:24	TXERR	The current value of the Tx Error Counter.	0	X

Πίνακας 23: Καταχωρητής CANGSR [5]

Ο καταχωρητής **CANICR** περιλαμβάνει πληροφορίες για όλα τα συμβάντα που έχουν συμβεί στο δίαυλο και είναι μόνο για ανάγνωση. Κάποια από τα bits του γίνονται μονάδες όταν έχουν συμβεί τα αντίστοιχα σφάλματα και μπορούν να προκαλέσουν διακοπή, εφόσον έχει γίνει η ανάλογη ρύθμιση του καταχωρητή CANIER. Όλα τα bits του καταχωρητή αυτού επιστρέφουν στην τιμή μηδέν όταν γίνει ανάγνωση είτε ολόκληρου είτε μέρους του καταχωρητή.

CANICR	Name	Function	Reset Value	RM Set
0	RI	1: Receive Interrupt -- this bit is set whenever the RBS bit in CANSR and the RIE bit in CANIER are both 1, indicating that a received message is available.=.	0	0
1	TI1	1: Transmit Interrupt 1 -- this bit is set when the TBS1 bit in CANSR goes from 0 to 1, indicating that Transmit buffer 1 is available, and the TIE1 bit in CANIER is 1.	0	0
2	EI	1: Error Warning Interrupt -- this bit is set on every change (set or clear) of the Error Status or Bus Status bit in CANSR, if the EIE bit in CAN is 1 at the time of the change.	0	X
3	DOI	1: Data Overrun Interrupt -- this bit is set when the DOS bit in CANSR goes from 0 to 1, if the DOIE bit in CANIE is 1.	0	0
4	WUI	1: Wake-Up Interrupt: this bit is set if the CAN controller is sleeping and bus activity is detected, if the WUIE bit in CANIE is 1.	0	0
5	EPI	1: Error Passive Interrupt -- this bit is set if the EPIE bit in CANIE is 1, and the CAN controller switches between Error Passive and Error Active mode in either direction.	0	0
6	ALI	1: Arbitration Lost Interrupt -- this bit is set if the ALIE bit in CANIE is 1, and the CAN controller loses arbitration while attempting to transmit.	0	0
7	BEI	1: Bus Error Interrupt -- this bit is set if the BEIE bit in CANIE is 1, and the CAN controller detects an error on the bus.	0	X
8	IDI	1: ID Ready Interrupt -- this bit is set if the IDIE bit in CANIE is 1, and a CAN Identifier has been received.	0	0
9	TI2	1: Transmit Interrupt 2 -- this bit is set when the TBS2 bit in CANSR goes from 0 to 1, indicating that Transmit buffer 2 is available, and the TIE2 bit in CANIER is 1.	0	0
10	TI3	1: Transmit Interrupt 1 -- this bit is set when the TBS3 bit in CANSR goes from 0 to 1, indicating that Transmit buffer 3 is available, and the TIE3 bit in CANIER is 1.	0	0
20:16	ERRBIT	Error Code Capture: when the CAN controller detects a bus error, the location of the error within the frame is captured in this field. The value reflects an internal state variable. 00010: ID28:21 / 01010: Data field / 10011: dominant OK bits 00011: Start of Frame / 01011: DLC / 10110: Passive error flag 00100: SRTR Bit / 01100: RTR bit / 10111: Error delimiter 00101: IDE Bit / 01101: Reserved Bit 1 / 11000: CRC delimiter 00110: ID20:18 / 01110: ID4:0 / 11001: Ack slot 00111: ID17:13 / 01111: ID12:5 / 11010: End of Frame 01000: CRC / 10001: Active Error flag / 11011: Ack delimiter 01001: Reserved Bit 0 / 10010: Intermission / 11100: Overload flag Reading this byte enables another Bus Error Interrupt.	0	X
21	ERRDIR	When the CAN controller detects a bus error, the direction of the current bit is captured in this bit. 1=receiving, 0=transmitting.	0	X
23:22	ERRC	When the CAN controller detects a bus error, the type of error is captured in this field: 00=bit error, 01=Form error, 10=Stuff error, 11=other error.	0	X
28:24	ALCBIT	Each time arbitration is lost while trying to send on the CAN, the bit number within the frame is captured into this field. 0 indicates arbitration loss in the first (MS) bit of the Identifier … 31 indicates loss in the RTR bit of an extended frame. After this byte is read, the ALI bit is cleared and a new Arbitration Lost interrupt can occur.	0	X

Πίνακας 24: Καταχωρητής CANICR [5]

Ο καταχωρητής **CANIER** συνδέεται άμεσα με τον προηγούμενο και όταν κάποια bits του είναι μονάδα, τότε η θέση του αντίστοιχου bit του CANICR σε μονάδα προκαλεί διακοπή.

CANIER	Name	Function	Reset Value	RM Set
0	RIE	Receiver Interrupt Enable.	0	X
1	TIE1	Transmit Interrupt Enable (1)	0	X
2	EIE	Error Warning Interrupt Enable	0	X
3	DOIE	Data Overrun Interrupt Enable	0	X
4	WUIE	Wake-Up Interrupt Enable	0	X
5	EPIE	Error Passive Interrupt Enable	0	X
6	ALIE	Arbitration Lost Interrupt Enable	0	X
7	BEIE	Bus Error Interrupt Enable	0	X
8	IDIE	ID Ready Interrupt Enable	0	X
9	TIE2	Transmit Interrupt Enable (2)	0	X
10	TIE3	Transmit Interrupt Enable (3)	0	X

Πίνακας 25: Καταχωρητής CANIER [5]

Ο έλεγχος του χρονισμού του ελεγκτή CAN γίνεται μέσω του καταχωρητή **CANBTR**. Αυτός, όπως και όλοι υπόλοιποι καταχωρητές που ρυθμίζουν τη λειτουργία του ελεγκτή μπορεί να εγγραφεί μόνο όταν ο ελεγκτής βρίσκεται σε κατάσταση επανεκκίνησης, δηλαδή το bit RM του CANMOD είναι μονάδα. Η λειτουργίες του καταχωρητή δίνονται στον παρακάτω πίνακα.

CANBTR	Name	Function	Reset Value	RM Set
0:9	BRP	Baud Rate Prescaler. The VPB clock is divided by (this value plus one) to produce the CAN clock.	0	X
15:14	SJW	The Synchronization Jump Width is (this value plus one) CAN clocks.	0	X
19:16	TSEG1	The delay from the nominal Sync point to the sample point is (this value plus one) CAN clocks.	1100	X
22:20	TSEG2	The delay from the sample point to the next nominal sync point is (this value plus one) CAN clocks. The nominal CAN bit time is (this value plus the value in TSEG1 plus 3) CAN clocks.	001	X
23	SAM	1: the bus is sampled 3 times (recommended for low to medium speed buses) 0: the bus is sampled once (recommended for high speed buses)	0	X

Πίνακας 26: Καταχωρητής CANBTR [5]

Οι ελεγκτές CAN του LPC2129 έχουν τη δυνατότητα να προκαλούν μια διακοπή όταν το πλήθος των σφαλμάτων κατά τη λήψη ή την αποστολή μηνυμάτων περάσει κάποιο καθορισμένο όριο. Το όριο αυτό καθορίζεται από την τιμή του καταχωρητή **CANEWL** ο οποίος μπορεί να εγγραφεί όταν ο ελεγκτής είναι σε κατάσταση επανεκκίνησης.

CANEWL	Name	Function	Reset Value	RM Set
7:0	EWL	During CAN operation, this value is compared to both the Tx and Rx Error Counters. If either of these counter matches this value, the Error Status (ES) bit in CANSR is set.	96_{10}=0x60	X

Πίνακας 27: Καταχωρητής CANEWL [5]

Ο καταχωρητής **CANSR**, που είναι μόνο για ανάγνωση, περιλαμβάνει πληροφορίες σχετικά με την κατάσταση των τριών buffers αποστολής (Tx Buffers), ενώ τα υπόλοιπα bits που δεν σχετίζονται με τους buffers αυτούς έχουν την τιμή που έχουν τα αντίστοιχα bits του κεντρικού καταχωρητή κατάστασης GSR.

CANSR	Name	Function	Reset Value	RM Set
0, 8, 16	RBS	These bits are identical to the RSB bit in the GSR.	0	0
1, 9, 17	DOS	These bits are identical to the DOS bit in the GSR.	0	0
2, 10, 18	TBS1, TBS2, TBS3	1: software may write a message into the CANTFI, CANTID, CANTDA, and CANTDB registers for this Tx Buffer. 0: software should not write to any of the CANTFI, CANTID, CANTDA, and CANTDB registers for this Tx Buffer.	1	X
3, 11, 19	TCS1, TCS2, TCS3	1: The previously requested transmission for this Tx Buffer has been successfully completed. 0: The previously requested transmission for this Tx Buffer is not complete.	1	0
4, 12, 20	RS	These bits are identical to the RS bit in the GSR.	0	0
5, 13, 21	TS1, TS2, TS3	1: The CAN Controller is transmitting a message from this Tx Buffer.	0	0
6, 14, 22	ES	These bits are identical to the ES bit in the GSR.	0	0
7, 15, 23	BS	These bits are identical to the BS bit in the GSR.	0	0

Πίνακας 28: Καταχωρητής CANSR [5]

Ο καταχωρητής **CANRFS** περιέχει τα χαρακτηριστικά του τελευταίου μηνύματος που έχει ληφθεί. Σε κανονική λειτουργία είναι μόνο για ανάγνωση αλλά μπορεί να εγγραφεί σε κατάσταση επανεκκίνησης για δοκιμαστικούς σκοπούς. Τα περιεχόμενά του περιγράφονται στον πίνακα.

CANRFS	Name	Function	Reset Value	RM Set
9:0	ID Index	If the BP bit (below) is 0, this value is the zero-based number of the Lookup Table RAM entry at which the Acceptance Filter matched the received Identifier. Disabled entries in the Standard tables are included in this numbering, but will not be matched. See the section "Examples of Acceptance Filter Tables and ID Index Values" on page 209 for examples of ID Index values.	0	X
10	BP	If this bit is 1, the current message was received in AF Bypass mode, and the ID Index field (above) is meaningless.	0	X
19:16	DLC	The field contains the Data Length Code (DLC) field of the current received message. When RTR=0, this is related to the number of data bytes available in the CANRDA and CANRDB registers as follows: 0000-0111 = 0 to 7 bytes 1000-1111 = 8 bytes With RTR=1, this value indicates the number of data bytes requested to be sent back, with the same encoding.	0	X
30	RTR	This bit contains the Remote Transmission Request bit of the current received message. 0 indicates a Data Frame, in which (if DLC is non-zero) data can be read from the CANRDA and possibly the CANRDB registers. 1 indicates a Remote frame, in which case the DLC value identifies the number of data bytes requested to be sent using the same Identifier.	0	X
31	FF	A 0 in this bit indicates that the current received message included an 11-bit Identifier, while a 1 indicates a 29-bit Identifier. This affects the contents of the CANid register described below.	0	X

Πίνακας 29: Καταχωρητής CANRFS [5]

Το αναγνωριστικό του τελευταίου μηνύματος που ελήφθει τοποθετείται στον καταχωρητή **CANRID** και είναι μόνο για ανάγνωση εκτός αν ο ελεγκτής βρίσκεται σε κατάσταση επανεκκίνησης οπότε μπορεί να εγγραφεί για δοκιμαστικούς σκοπούς. Ανάλογα με την κατάσταση του bit FF του παραπάνω καταχωρητή CANRFS, το αναγνωριστικό έχει μήκος είτε 11 είτε 29 bits και αποθηκεύεται με μια από τις παρακάτω μορφές.

Όταν το bit FF=0:

CANRID	Name	Function	Reset Value	RM Set
10:0	ID	The 11-bit Identifier field of the current received message. In CAN 2.0A, these bits are called ID10-0, while in CAN 2.0B they're called ID29-18.	0	X

Όταν το bit FF=1:

CANRID	Name	Function	Reset Value	RM Set
28:0	ID	The 29-bit Identifier field of the current received message. In CAN 2.0B these bits are called ID29-0.	0	X

Πίνακας 30: Καταχωρητής CANRID [5]

Τα περιεχόμενα του τελευταίου μηνύματος που έχει ληφθεί τοποθετούνται στους καταχωρητές **CANRDA** και **CANRDB** με τον πρώτο να έχει τα πρώτα 4 και το δεύτερο τα υπόλοιπα 4 bytes. Όπως και οι υπόλοιποι καταχωρητές του συστήματος λήψης, είναι μόνο για ανάγνωση εκτός από δοκιμαστικές περιπτώσεις.

CANRDA	Name	Function	Reset Value	RM Set
7:0	Data 1	If the DLC field in CANRFS >= 0001, this contains the first Data byte of the current received message.	0	X
15:8	Data 2	If the DLC field in CANRFS >= 0010, this contains the second Data byte of the current received message.	0	X
23:16	Data 3	If the DLC field in CANRFS >= 0011, this contains the third Data byte of the current received message.	0	X
31:24	Data 4	If the DLC field in CANRFS >= 0100, this contains the fourth Data byte of the current received message.	0	X

CANRDB	Name	Function	Reset Value	RM Set
7:0	Data 5	If the DLC field in CANRFS >= 0101, this contains the 5th Data byte of the current received message.	0	X
15:8	Data 6	If the DLC field in CANRFS >= 0110, this contains the 6th Data byte of the current received message.	0	X
23:16	Data 7	If the DLC field in CANRFS >= 0111, this contains the 7th Data byte of the current received message.	0	X
31:24	Data 8	If the DLC field in CANRFS >= 1000, this contains the 8th Data byte of the current received message.	0	X

Πίνακας 31: Καταχωρητές CANRDA και CANRDB [5]

Οι καταχωρητές του συστήματος αποστολής μηνυμάτων μπορούν να εγγραφούν όταν το bit TBS του καταχωρητή CANSR είναι μονάδα δηλαδή όταν ο ελεγκτής είναι έτοιμος για αποστολή. Τα bits των καταχωρητών αυτών που δεν αναφέρονται στους παρακάτω πίνακες έχουν τιμή 0 κατά την ανάγνωση και πρέπει να γράφονται σε τιμή 0. Η μορφή του μηνύματος που πρόκειται να σταλεί καθορίζεται από τον καταχωρητή **CANTFI** για κάθε έναν από τους buffers αποστολής σύμφωνα με τον παρακάτω πίνακα:

CANTFI	Name	Function	Reset Value	RM Set
7:0	PRIO	If the TPM bit in the CANMOD register is 1, enabled Tx Buffers contend for the right to send their messages based on this field. The lowest binary value has priority.		
19:16	DLC	This value is sent in the DLC field of the next transmit message. In addition, if RTR=0, this value controls the number of Data bytes sent in the next transmit message, from the CANTDA and CANTDB registers: 0000-0111 = 0-7 bytes 1xxx = 8 bytes	0	X
30	RTR	This value is sent in the RTR bit of the next transmit message. If this bit is 0, the number of data bytes called out by the DLC field are sent from the CANTDA and CANTDB registers. If it's 1, a Remote Frame is sent, containing a request for that number of bytes.	0	X
31	FF	If this bit is 0, the next transmit message will be sent with an 11-bit Identifier, while if it's 1, the message will be sent with a 29-bit Identifier.	0	X

Πίνακας 32: Καταχωρητής CANTFI [5]

Αντίστοιχα με το σύστημα λήψης, το αναγνωριστικό του μηνύματος που πρόκειται να σταλεί γράφεται στον καταχωρητή **CANTID** και έχει μήκος ανάλογα με την τιμή που έχει οριστεί στο bit FF του CANTFI. Οι δύο μορφές είναι:

CANTID	Name	Function	Reset Value	RM Set
10:0	ID	The 11-bit Identifier to be sent in the next transmit message.	0	X
28:0	ID	The 29-bit Identifier to be sent in the next transmit message.	0	X

Πίνακας 33: Καταχωρητής CANTID [5]

Τα πρώτα 4 bytes και τα επόμενα 4 bytes δεδομένων του μηνύματος που πρόκειται να σταλεί γράφονται αντίστοιχα στους καταχωρητές **CANTDA** και **CANTDB**.

CANTDA	Name	Function	Reset Value	RM Set
7:0	Data 1	If RTR=0 and DLC >= 0001 in the corresponding CANTFI, this byte is sent as the first Data byte of the next transmit message.	0	X
15:8	Data 2	If RTR=0 and DLC >= 0010 in the corresponding CANTFI, this byte is sent as the 2nd Data byte of the next transmit message.	0	X
23:16	Data 3	If RTR=0 and DLC >= 0011 in the corresponding CANTFI, this byte is sent as the 3rd Data byte of the next transmit message.	0	X
31:24	Data 4	If RTR=0 and DLC >= 0100 in the corresponding CANTFI, this byte is sent as the 4th Data byte of the next transmit message.	0	X

CANTDB	Name	Function	Reset Value	RM Set
7:0	Data 5	If RTR=0 and DLC >= 0101 in the corresponding CANTFI, this byte is sent as the 5th Data byte of the next transmit message.	0	X
15:8	Data 6	If RTR=0 and DLC >= 0110 in the corresponding CANTFI, this byte is sent as the 6th Data byte of the next transmit message.	0	X
23:16	Data 7	If RTR=0 and DLC >= 0111 in the corresponding CANTFI, this byte is sent as the 7th Data byte of the next transmit message.	0	X
31:24	Data 8	If RTR=0 and DLC >= 1000 in the corresponding CANTFI, this byte is sent as the 8th Data byte of the next transmit message.	0	X

Πίνακας 34: Καταχωρητές CANTDA και CANTDB [5]

Σε αυτό το σημείο πρέπει να σημειωθεί ότι με βάση το τελευταίο errata sheet του LPC2129 με ημερομηνία έκδοσης 17 Μαΐου 2006, **το σύστημα των πολλαπλών buffers αποστολής δεν λειτουργεί**. Η κατασκευάστρια Philips προτείνει τη χρήση μόνο ενός από τους buffers αποστολής σε όλο το μήκος του προγράμματος.

Από τους κεντρικούς καταχωρητές του συστήματος CAN, τρεις αποτελούν την ομάδα καταχωρητών κατάστασης ενώ οι υπόλοιποι, που θα αναλυθούν στην επόμενη ενότητα, σχετίζονται με το σύστημα φιλτραρίσματος και αποδοχής των μηνυμάτων. Από τους καταχωρητές κατάστασης, ο **CANTxSR** περιλαμβάνει πληροφορίες για την αποστολή μηνυμάτων, ο **CANRxSR** για τη λήψη τους και ο **CANMSR** άλλες πληροφορίες. Τα περιεχόμενά τους δίνονται στους τρεις πίνακες που ακολουθούν.

CANTxSR	Name	Function	Reset Value
3:0	TS4:1	1: the CAN controller is sending a message (same as TS in the CANGSR) TS4:3 are available in LPC2294 only. In other parts these bits are reserved.	0
7:4	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
11:8	TBS4:1	1: all 3 Tx Buffers are available to the CPU (same as TBS in CANGSR) TBS4:3 are available in LPC2294 only. In other parts these bits are reserved.	all 1
15:12	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
19:16	TCS4:1	1: all requested transmissions have been completed successfully (same as TCS in CANGSR) TCS4:3 are available in LPC2294 only. In other parts these bits are reserved.	all 1
31:20	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Πίνακας 35: Καταχωρητής CANTxSR [5]

CANRxSR	Name	Function	Reset Value
3:0	RS4:1	1: the CAN controller is receiving a message (same as RS in CANGSR) RS4:3 are available in LPC2294 only. In other parts these bits are reserved.	0
7:4	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
11:8	RBS4:1	1: a received message is available in the CAN controller (same as RBS in CANGSR) RBS4:3 are available in LPC2294 only. In other parts these bits are reserved.	0
15:12	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
19:16	DOS4:1	1: a message was lost because the preceding message to this CAN controller was not read out quickly enough (same as DOS in CANGSR) DOS4:3 are available in LPC2294 only. In other parts these bits are reserved.	0
31:20	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Πίνακας 36: Καταχωρητής CANRxSR [5]

CANMSR	Name	Function	Reset Value
3:0	ES4:1	1: one or both of the Tx and Rx Error Counters has reached the limit set in the EWL register (same as ES in CANGSR) ES4:3 are available in LPC2294 only. In other parts these bits are reserved.	0
7:4	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
11:8	BS4:1	1: the CAN controller is currently involved in bus activities (same as BS in CANGSR) BS4:3 are available in LPC2294 only. In other parts these bits are reserved.	0
31:12	Reserved	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Πίνακας 37: Καταχωρητής CANMSR [5]

Λειτουργία των ελεγκτών CAN του LPC2129

Όσον αφορά το χειρισμό των σφαλμάτων, ο ελεγκτής CAN μπορεί να βρεθεί σε τρεις καταστάσεις ανάλογα με το πλήθος των σφαλμάτων που έχουν αναγνωριστεί. Η πρώτη κατάσταση θεωρείται κανονική και συμβαίνει για όσο διάστημα το πλήθος των σφαλμάτων σε οποιονδήποτε από τους δύο μετρητές (σφάλματα κατά την αποστολή-TXERR ή τη λήψη-RXERR) είναι μικρότερο από το όριο που έχει τεθεί στον καταχωρητή CANEWL. Όταν το όριο αυτό ξεπεραστεί, ο ελεγκτής εισέρχεται σε κατάσταση «error passive» και ενεργοποιείται η αντίστοιχη διακοπή, αν επιτρέπεται.

Η τρίτη κατάσταση λέγεται «bus-off» και ενεργοποιείται όταν ο μετρητής σφαλμάτων αποστολής TXERR πάρει τιμή ίση με 255. Σε αυτήν την κατάσταση, το bit BS στον καταχωρητή CANSR τίθεται μονάδα, ενεργοποιείται η αντίστοιχη διακοπή, αν επιτρέπεται, και ο ελεγκτής μπαίνει σε κατάσταση επανεκκίνησης με το bit RM του CANMOD να είναι μονάδα. Στην κατάσταση αυτή ο κόμβος ουσιαστικά παύει να επικοινωνεί με το δίαυλο ώστε, αν ο συγκεκριμένος κόμβος είναι που προκαλεί τα σφάλματα, ο δίαυλος να συνεχίσει να λειτουργεί κανονικά για τους υπόλοιπους κόμβους που είναι συνδεδεμένοι σε αυτόν. Κατά την είσοδο στην κατάσταση «bus-off», ο μετρητής σφλαμάτων λήψης μηδενίζεται και ο μετρητής σφαλμάτων αποστολής παίρνει αυτόματα την τιμή 127 και ελαττώνεται κάθε φορά που αναγνωρίζεται ότι ο δίαυλος είναι ελεύθερος, δηλαδή εμφανίζονται 11 διαδοχικά «recessive» bits. Είσοδος στην κατάσταση «bus-off» μπορεί να προκληθεί και από το πρόγραμμα, εφόσον ο ελεγκτής είναι σε επανεκκίνηση και ο μετρητής σφαλμάτων αποστολής τεθεί ίσος με 255. Για την έξοδο από την κατάσταση «bus-off», το πρόγραμμα θα πρέπει να γράψει οποιαδήποτε

τιμή μεταξύ 0 και 254 στο μετρητή σφαλμάτων αποστολής και να θέσει το bit RM του CANMOD ίσο με μηδενικό. Στη συνέχεια, αρκεί ο δίαυλος να βρεθεί ελεύθερος για να συνεχιστεί η κανονική λειτουργία του κόμβου.

Ο ελεγκτής CAN έχει τη δυνατότητα εισόδου σε κατάσταση αναμονής («Sleep mode») με κατάλληλο χειρισμό από το πρόγραμμα και εφόσον δεν στέλνει ή δέχεται μήνυμα καθώς και δεν εκκρεμεί κάποια διακοπή. Επανέρχεται σε κανονική λειτουργία είτε όταν αναγνωρίσει κάποια δραστηριότητα στο δίαυλο, δηλαδή ένα «dominant» bit, είτε όταν το πρόγραμμα τον ενεργοποιήσει. Αν βρίσκεται σε κατάσταση λειτουργίας, το πρώτο μήνυμα που θα δεχθεί θα τον αφυπνίσει αλλά δεν θα το λάβει, παρά μόνο αφότου αναγνωρίσει ότι ο δίαυλος είναι ελεύθερος.

Σύμφωνα με το errata sheet του επεξεργαστή, το σύστημα εξοικονόμησης ενέργειας αντιμετωπίζει μερικά προβλήματα σχετικά με τους ελεγκτές CAN. Πρώτον, οι ελεγκτές CAN δεν μπορούν να αφυπνίσουν ολόκληρο τον επεξεργαστή αν αυτός βρίσκεται σε κατάσταση μη-λειτουργίας (Power Down mode). Δεύτερον, η αφύπνιση του ίδιου του ελεγκτή από την κατάσταση αναμονή (Sleep mode) δεν μπορεί να γίνει από το πρόγραμμα αλλά μόνο από εξωτερική δραστηριότητα. Τα παραπάνω υπαγορεύουν τη μη χρήση των δύο αυτών λειτουργιών εξοικονόμησης ενέγρειας του επεξεργαστή σε αυτήν την εργασία.

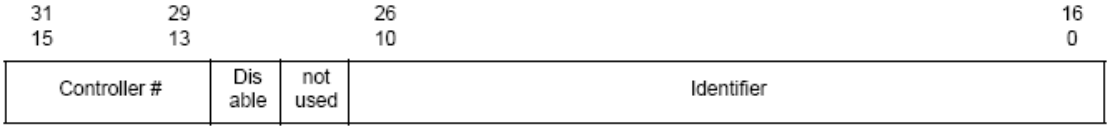
Φιλτράρισμα

Μια πολύ σημαντική δυνατότητα που δίνουν οι ελεγκτές CAN του LPC2129 είναι το φιλτράρισμα των μηνυμάτων σε επίπεδο υλικού με βάση το αναγνωριστικό τους, ώστε να μην επιβαρύνεται ο επεξεργαστής με την επεξεργασία όλων των μηνυμάτων που μεταδίδονται στο δίαυλο. Το σύστημα φιλτραρίσματος αποτελείται από έναν κεντρικό καταχωρητή ελέγχου, ο οποίος ρυθμίζει αν ο ελεγκτής δέχεται μηνύματα, αν το φίλτρο είναι ενεργό ή παρακάμπτεται ώστε να περνούν όλα τα ληφθέντα μηνύματα και αν το σύστημα λειτουργεί σε κατάσταση «FullCAN». Εκτός του κεντρικού αυτού καταχωρητή, το σύστημα φιλτραρίσματος αποτελείται από πέντε πίνακες, οι τρεις από τους οποίους αφορούν μηνύματα με βασική δομή ενώ οι άλλοι δύο με εκτεταμένη.

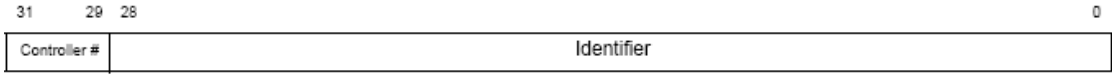
Στην κανονική κατάσταση λειτουργίας, δηλαδή όχι «FullCAN», το σύστημα απλά ελέγχει αν το μήνυμα που ελήφθει γίνεται δεκτό και το διαθέτει προς αποθήκευση και επεξεργασία στο πρόγραμμα. Στην κατάσταση «FullCAN», το σύστημα το ίδιο φροντίζει να αποθηκεύσει το μήνυμα, η επεξεργασία του οποίου γίνεται στη συνέχεια από το

πρόγραμμα. Η λειτουργία αυτή υπάρχει μόνο για μηνύματα βασικής δομής και ελαχιστοποιεί την επιβάρυνση του κεντρικού επεξεργαστή αλλά αντιμετωπίζει κάποια προβλήματα που περιγράφονται στο errata sheet του LPC2129, οπότε δεν θα χρησιμοποιηθεί στη συγκεκριμένη εφαρμογή.

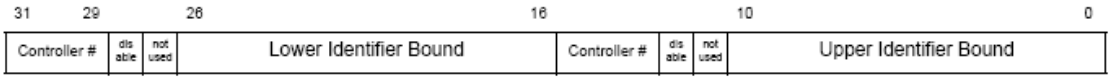
Οι πέντε πίνακες που αναφέρθηκαν περιέχουν τα κριτήρια με βάση τα οποία γίνεται η αποδοχή ή η απόρριψη ενός μηνύματος. Ο πρώτος αναφέρεται στη λειτουργία «FullCAN» και δεν θα απασχολήσει την ανάλυση. Οι δύο επόμενοι αναφέρονται σε μηνύματα βασικής και οι δύο τελευταίοι σε εκτεταμένης δομής. Συγκεκριμένα, ο πρώτος από το κάθε ζευγάρι περιέχει τα αναγνωριστικά που θεωρούνται δεκτά και ο δεύτερος τις περιοχές αναγνωριστικών τα οποία θεωρούνται δεκτά. Η μορφή με την οποία αποθηκεύονται τα αναγνωριστικά στον πρώτο πίνακα κάθε ζευγαριού είναι:



όταν πρόκειται για βασική δομή με αναγνωριστικό μήκους 11 bits ή:



όταν πρόκειται για εκτεταμένη δομή με αναγνωριστικό μήκους 29 bits. Οι περιοχές αναγνωριστικών που περιλαμβάνονται στο δεύτερο πίνακα κάθε ζευγαριού ορίζονται αποθηκεύοντας το αναγνωριστικό αρχής και τέλους της περιοχής, στη μορφή:



όταν πρόκειται για βασική δομή, ενώ στην εκτεταμένη η αποθήκευση γίνεται όπως και στον πρώτο πίνακα με τη διαφορά ότι το σύστημα θεωρεί τις περιττές γραμμές ως κάτω όρια και τις άρτιες ως άνω. Το bit «Disable» χρησιμοποιείται για την απενεργοποίηση ενός συγκεκριμένου αναγνωριστικού κατά την εκτέλεση του προγράμματος.

Όταν λαμβάνεται ένα μήνυμα και το φιλτράρισμα είναι ενεργοποιημένο, το σύστημα ελέγχει το μήκος του αναγνωριστικού για να αποφασίσει αν πρόκειται για

μήνυμα βασικής ή εκτεταμένης δομής. Στη συνέχεια ελέγχει τον πρώτο πίνακα του αντίστοιχου ζευγαριού και αν βρει ότι το αναγνωριστικό του μηνύματος που ελήφθει είναι ίδιο με κάποιο από τα αναγνωριστικά που περιλαμβάνονται στον πίνακα, ειδοποιεί τον ελεγκτή μεταφέροντας τα περιεχόμενα του μηνύματος στους αντίστοιχους καταχωρητές και εγγράφει στο πεδίο ID Index του καταχωρητή CANRFS ένα δείκτη ο οποίος είναι ο αριθμός της γραμμής του πίνακα στην οποία βρέθηκε το συγκεκριμένο αναγνωριστικό. Αν το σύστημα δεν βρει το αναγνωριστικό του μηνύματος ανάμεσα στα αναγνωριστικά του πρώτου πίνακα, ελέγχει το δεύτερο για να δει αν το αναγνωριστικό αυτό βρίσκεται εντός των περιοχών που έχουν οριστεί. Αν έχει επιτυχία, ακολουθεί την ίδια διαδικασία με προηγουμένως με το δείκτη αυτή τη φορά να είναι ο αριθμός της αντίστοιχης γραμμής του δεύτερου πίνακα αυξημένος κατά το πλήθος των γραμμών του πρώτου πίνακα. Με τον τρόπο αυτό, αποφεύγεται η σύγχυση για το αν το αναγνωριστικό του μηνύματος που ελήφθει βρέθηκε στον πρώτο ή στο δεύτερο πίνακα. Τέλος, αν δεν υπάρξει επιτυχία σε κανέναν από τους δύο πίνακες, το μήνυμα απορρίπτεται.

Ο κεντρικός καταχωρητής του συστήματος φιλτραρίσματος είναι ο **AFMR** και οι λειτουργίες του αναλύονται στον παρακάτω πίνακα:

AFMR	Name	Function	Reset Value
0	AccOff	1: if AccBP is 0, the Acceptance Filter is not operational. All Rx messages on all CAN buses are ignored.	1
1	AccBP	1: all Rx messages are accepted on enabled CAN controllers. Software must set this bit before modifying the contents of any of the registers described below, and before modifying the contents of Lookup Table RAM in any way other than setting or clearing Disable bits in Standard Identifier entries. When both this bit and AccOff are 0, the Acceptance filter operates to screen received CAN Identifiers.	0
2	eFCAN	1: the Acceptance Filter itself will take care of receiving and storing messages for selected Standard ID values on selected CAN buses. See "FullCAN Mode" on page 210. 0: software must read all messages for all enabled IDs on all enabled CAN buses, from the receiving CAN controllers.	0

Πίνακας 38: Καταχωρητής AFMR [5]

Οι πίνακες του συστήματος τοποθετούνται στη μνήμη RAM και οι διεθύνσεις τους καθορίζονται από τα περιεχόμενα των καταχωρητών **SFF_sa**, **SFF_GRP_sa**, **EFF_sa**, **EFF_GRP_sa** και **ENDofTable**, οι οποίοι ορίζουν τη διεύθυνση από την οποία ξεκινάνε οι πίνακες αυτοί. Ο τελευταίος καταχωρητής περιέχει τη διεύθυνση στην οποία τελειώνει η τελευταία γραμμή του τελευταίου πίνακα.

SFF_sa	Name	Function	Reset Value
10:2		The start address of the table of individual Standard Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the SFF_GRP_sa register described below. For compatibility with possible future devices, please write zeroes in bits 31:11 and 1:0 of this register. If the eFCAN bit in the AFMR is 1, this value also indicates the size of the table of Standard IDs which the Acceptance Filter will search and (if found) automatically store received messages in Acceptance Filter RAM.	0

SFF_GRP_sa	Name	Function	Reset Value
11:2		The start address of the table of grouped Standard Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the EFF_sa register described below. The largest value that should be written to this register is 0x800, when only the Standard Individual table is used, and the last word (address 0x7FC) in AF Lookup Table RAM is used. For compatibility with possible future devices, please write zeroes in bits 31:12 and 1:0 of this register.	0

EFF_sa	Name	Function	Reset Value
10:2		The start address of the table of individual Extended Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the EFF_GRP_sa register described below. The largest value that should be written to this register is 0x800, when both Extended Tables are empty and the last word (address 0x7FC) in AF Lookup Table RAM is used. For compatibility with possible future devices, please write zeroes in bits 31:11 and 1:0 of this register.	0

EFF_GRP_sa	Name	Function	Reset Value
11:2		The start address of the table of grouped Extended Identifiers in AF Lookup RAM. If the table is empty, write the same value in this register and the ENDofTable register described below. The largest value that should be written to this register is 0x800, when this table is empty and the last word (address 0x7FC) in AF Lookup Table RAM is used. For compatibility with possible future devices, please write zeroes in bits 31:12 and 1:0 of this register.	0

ENDofTable	Name	Function	Reset Value
11:2		The address above the last active address in the last active AF table. For compatibility with possible future devices, please write zeroes in bits 31:12 and 1:0 of this register. If the eFCAN bit in the AFMR is 0, the largest value that should be written to this register is 0x800, which allows the last word (address 0x7FC) in AF Lookup Table RAM to be used. If the eFCAN bit in the AFMR is 1, this value marks the start of the area of Acceptance Filter RAM, into which the Acceptance Filter will automatically receive messages for selected IDs on selected CAN buses. In this case, the maximum value that should be written to this register is 0x800 minus 6 times the value in SFF_sa. This allows 12 bytes of message storage between this address and the end of Acceptance Filter RAM, for each Standard ID that is specified between the start of Acceptance Filter RAM, and the next active AF table.	0

Πίνακας 39: Καταχωρητές SFF_sa, SFF_GRP_sa, EFF_sa, EFF_GRP_sa και ENDofTable [5]

Τέλος, το παραπάνω σύστημα έχει τη δυνατότητα να ελέγχει τις καταχωρήσεις των πινάκων με τα αναγνωριστικά και να αναγνωρίζει πιθανά συντακτικά λάθη, προκαλώντας διακοπή αν επιτρέπεται και δηλώνοντας το δείκτη της γραμμής στην οποία υπάρχει το πρόβλημα. Όπως αναφέρεται στο errata sheet, η δυνατότητα αυτή δεν λειτουργεί σωστά, επομένως δεν θα χρησιμοποιηθεί στην εφαρμογή.

Εκτός των προβλημάτων που αναφέρθηκαν παραπάνω, στο errata sheet του LPC2129 περιγράφονται και μερικές ακόμα δυσλειτουργίες που σχετίζονται με τους ελεγκτές CAN. Η μία εμφανίζεται κατά τη μετάβαση από κατάσταση επανεκκίνησης σε κατάσταση λειτουργίας και η άλλη όταν ο κόμβος που στέλνει ένα μήνυμα χάνει τη

διαιτησία αλλά δεν λαμβάνει σωστά το μήνυμα του κόμβου που κέρδισε τη διαιτησία. Για την αντιμετώπιση του πρώτου προβλήματος, όταν επιχειρείται η μετάβαση από κατάσταση επανεκκίνησης σε κατάσταση λειτουργίας πρέπει αρχικά να σταλεί ένα κενό μήνυμα με αναγνωριστικό 0x0 και έχοντας θέσει τα bits SRR και AT του καταχωρητή CANCMR σε μονάδα. Αυτό έχει σαν αποτέλεσμα το κενό αυτό μήνυμα να σταλεί μόνο μια φορά ακόμα και αν δεν επιβεβαιωθεί από κάποιον κόμβο. Για την αντιμετώπιση του δεύτερου προβλήματος, η Philips προτείνει τη χρήση της εντολής SRR αντί της TR στον καταχωρητή CANCMR για την αποστολή ενός μηνύματος, κίνηση που σημαίνει ότι κάθε μήνυμα που θα στέλνεται θα θεωρείται ταυτόχρονα ληφθέν από τον κόμβο που το έστειλε.

Χρονιστές

Ο LPC2129 διαθέτει δύο ενσωματωμένες συσκευές χρονισμού που έχουν σχεδιαστεί ώστε να μετρούν κύκλους του ρολογιού των περιφερειακών (pclk) και να προκαλούν διακοπές ή άλλες ενέργειες σε συγκεκριμένες τιμές των αντίστοιχων μετρητών με βάση τέσσερις καταχωρητές σύγκρισης. Επιπλέον, διαθέτουν εισόδους από την κατάσταση των οποίων μπορεί να επηρεαστεί με ορισμένο τρόπο ο κάθε χρονιστής καθώς και εξόδους των οποίων η συμπεριφορά μπορεί να ρυθμιστεί ανάλογα με την τιμή του μετρητή.

Πιο συγκεκριμένα, τα στοιχεία που αποτελούν κάθε χρονιστή ξεχωριστά είναι ένας μετρητής 32 bit που συνδυάζεται με έναν 32-bit καταχωρητή επιλογής κλίμακας, τέσσερις καταχωρητές που μπορούν να αποθηκεύσουν την τιμή του μετρητή σε μεταβάσεις του σήματος των εισόδων και να προκαλέσουν διακοπή, κατ' επιλογή, τέσσερις καταχωρητές σύγκρισης στις τιμές των οποίων όταν φτάσει ο μετρητής μπορούν να προκληθούν διακοπές, παύση ή επανεκκίνηση του μετρητή και τέσσερις εξόδους που συνεργάζονται με τους καταχωρητές σύγκρισης και σε περίπτωση σύμπτωσης της τιμής του μετρητή μπορούν να βρεθούν σε κατάσταση μονάδας, μηδενικού ή να μεταβάλλουν την προηγούμενη κατάστασή τους.

Οι καταχωρητές των δύο χρονιστών είναι πανομοιότυποι αλλά βρίσκονται σε διαφορετικό τμήμα διευθύνσεων μνήμης. Η ενεργοποίηση κάποιας από τις δυνατές διακοπές αντικατοπτρίζεται στον καταχωρητή **IR** (T0IR και T1IR για τον πρώτο και το δεύτερο χρονιστή αντίστοιχα), κάθε bit του οποίου αντιστοιχεί σε έναν τύπο διακοπής.

Ο καταχωρητής ελέγχου της λειτουργίας της συσκευής χρονισμού είναι ο **TCR**, η λειτουργία του οποίου δίνεται στον παρακάτω πίνακα:

TCR	Function	Description	Reset Value
0	Counter Enable	When one, the Timer Counter and Prescale Counter are enabled for counting. When zero, the counters are disabled.	0
1	Counter Reset	When one, the Timer Counter and the Prescale Counter are synchronously reset on the next positive edge of pclk. The counters remain reset until TCR[1] is returned to zero.	0

Πίνακας 40: Καταχωρητής TCR των χρονιστών [5]

Η τιμή του κεντρικού μετρητή βρίσκεται στον καταχωρητή **TC** και αυξάνεται κατά 1 κάθε φορά που ο μετρητής κλίμακας φτάνει στην τιμή που έχει οριστεί στον καταχωρητή επιλογής κλίμακας. Η τιμή του μετρητή κλίμακας βρίσκεται στον καταχωρητή **PC** και αυξάνεται κατά 1 σε κάθε παλμό του ρολογιού των περιφερειακών pclk. Όταν η τιμή του φτάσει την τιμή που έχει οριστεί στον καταχωρητή επιλογή κλίμακας **PR** ο κεντρικός μετρητής αυξάνεται κατά μία μονάδα και ο μετρητής κλίμακας επιστρέφει στο μηδέν. Αν ο κεντρικός μετρητής φτάσει στη μέγιστη τιμή του, τότε επιστρέφει στο μηδέν χωρίς να προκαλείται αυτόματα κάποια ενέργεια ή διακοπή.

Αν είναι επιθυμητή η εκτέλεση ενεργειών σε συγκεκριμένες τιμές του κεντρικού μετρητή, τότε οι τιμές αυτές αποθηκεύονται στους τέσσερις καταχωρητές σύγκρισης **MR0** – **MR3**. Ο καταχωρητής **MCR** καθορίζει τις ενέργειες που εκτελούνται όταν η τιμή του κεντρικού μετρητή TC συμπέσει με τις τιμές κάθε ενός από τους καταχωρητές σύγκρισης MR0 – MR3. Οι δυνατές ενέργειες είναι η πρόκληση διακοπής, η επανεκκίνηση του μετρητή ή η παύση της λειτουργίας του και καθορίζεται για κάθε καταχωρητή σύγκρισης ξεχωριστά.

Κάθε εξωτερική είσοδος της συσκευής χρονισμού αντιστοιχεί σε έναν από τους τέσσερις καταχωρητές αποθήκευσης **CR0** – **CR3**. Η χρήση των εξωτερικών εισόδων υλοποιείται μέσω του καταχωρητή ελέγχου **CCR**, ο οποίος καθορίζει αν η αποθήκευση της τιμής του κεντρικού μετρητή σε κάθε έναν από τους καταχωρητές CR0 – CR3 θα γίνεται σε θετική ή αρνητική ακμή οι και στις δύο ακμές στην είσοδο καθώς και αν την ίδια στιγμή θα προκαλείται διακοπή.

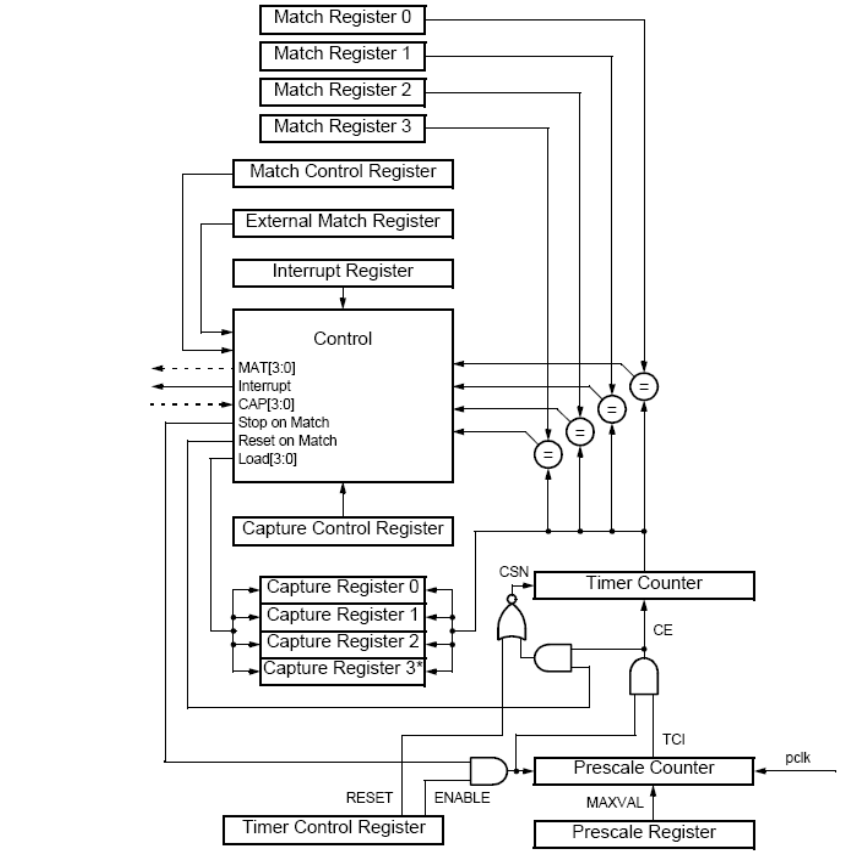
Ο έλεγχος των εξόδων του χρονιστή γίνεται μέσω του καταχωρητή **EMR**. Τα τέσσερα χαμηλότερα bits του καταχωρητή παίρνουν την τιμή που αντιστοιχεί στην κατάσταση των εξόδων, ανεξάρτητα από το αν είναι πράγματι ενεργοποιημένη η

αντίστοιχη έξοδος στους ακροδέκτες του μικροεπεξεργαστή. Η κατάσταση των εξόδων μεταβάλλεται όταν ο κεντρικός μετρητής λάβει την τιμή που είναι αποθηκευμένη στον αντίστοιχο καταχωρητή σύγκρισης MR0 – MR3. Τα επόμενα bits του καταχωρητή EMR καθορίζουν τη συμπεριφορά κάθε εξόδου ξεχωριστά, σύμφωνα με τον παρακάτω πίνακα:

EMR[11:10], EMR[9:8], EMR[7:6], or EMR[5:4]	Function
0 0	Do Nothing
0 1	Clear corresponding External Match output to 0 (LOW if pinned out)
1 0	Set corresponding External Match output to 1(HIGH if pinned out)
1 1	Toggle corresponding External Match output

Πίνακας 41: Έλεγχος λειτουργίας εξόδων χρονιστή [5]

Η αρχιτεκτονική καθενός από τους δύο χρονιστές φαίνεται στο παρακάτω μπλοκ διάγραμμα:



* Note that Capture Register 3 cannot be used on TIMER0

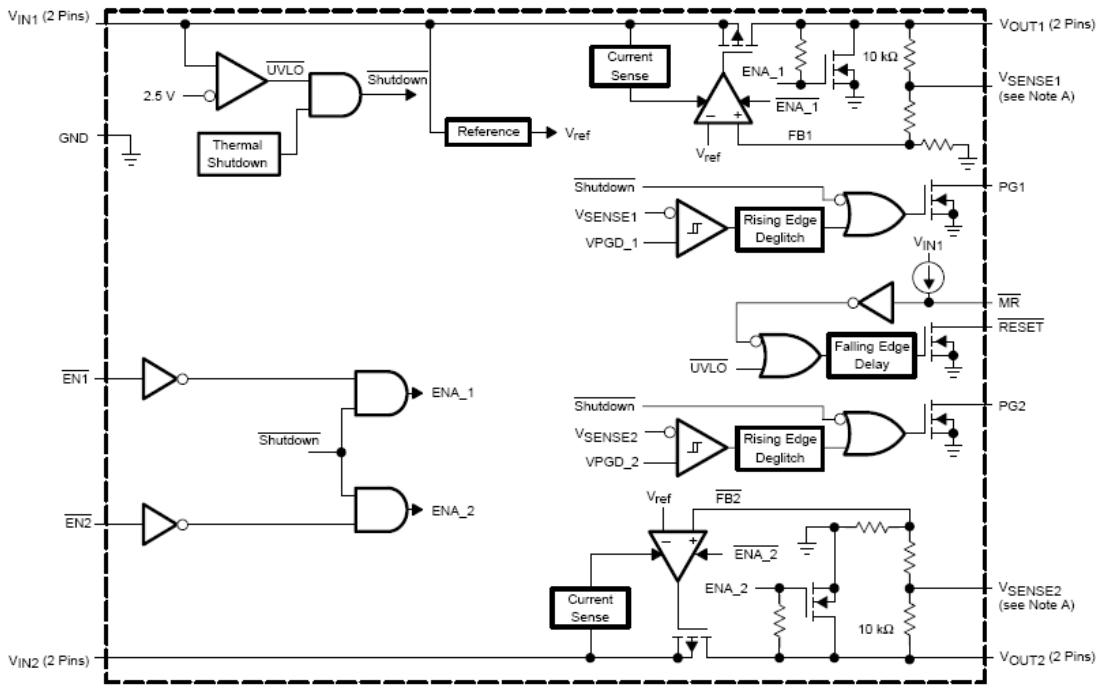
Εικόνα 16: Αρχιτεκτονική των χρονιστών [5]

ΡΥΘΜΙΣΤΗΣ ΤΑΣΗΣ TPS70251

Η λειτουργία του μικροεπεξεργαστή στην τυπωμένη πλακέτα απαιτεί την ύπαρξη δύο πηγών τάσης στα 3.3V και στα 1.8V για τα ενσωματωμένα περιφερειακά και την ΚΜΕ αντίστοιχα. Με δεδομένη την απαίτηση κατασκευής ενός κυκλώματος με μονή τροφοδοσία στο στάνταρ των 5V, τα επίπεδα τάσης αυτά θα πρέπει να παράγονται από κατάλληλη διάταξη. Το ρόλο αυτό αναλαμβάνει το ολοκληρωμένο TPS70251, το οποίο αποτελεί ένα διπλό ρυθμιστή τάσης.

Τα κύρια χαρακτηριστικά του περιλαμβάνουν δύο ανεξάρτητα συστήματα ρύθμισης τάσης με δυνατότητα απενεργοποίησης του καθενός από αυτά, μέγιστο ρεύμα εξόδου 500 mA για την πρώτη και 250 mA για τη δεύτερη, μικρές χρονικές υστερήσεις, μικρή διακύμανση της τάσης εξόδου μέσω συστήματος ανάδρασης, ελάχιστο ρεύμα απωλειών εξαιτίας της χρήσης τρανζίστορ PMOS στην έξοδο, χαμηλά επίπεδα θορύβου, συστήματα προστασίας της εξόδου σε διακυμάνσεις της τάσης εισόδου ή σε καταστάσεις υπερθέρμανσης και τέλος, σύστημα επανεκκίνησης ικανό για την παραγωγή των αντίστοιχων σημάτων ελέγχου της επανεκκίνησης των υπολοίπων ολοκληρωμένων της τυπωμένης πλακέτας.

Το μπλοκ διάγραμμα του ολοκληρωμένου είναι:



Εικόνα 17: Μπλοκ διάγραμμα του TPS70251 [7]

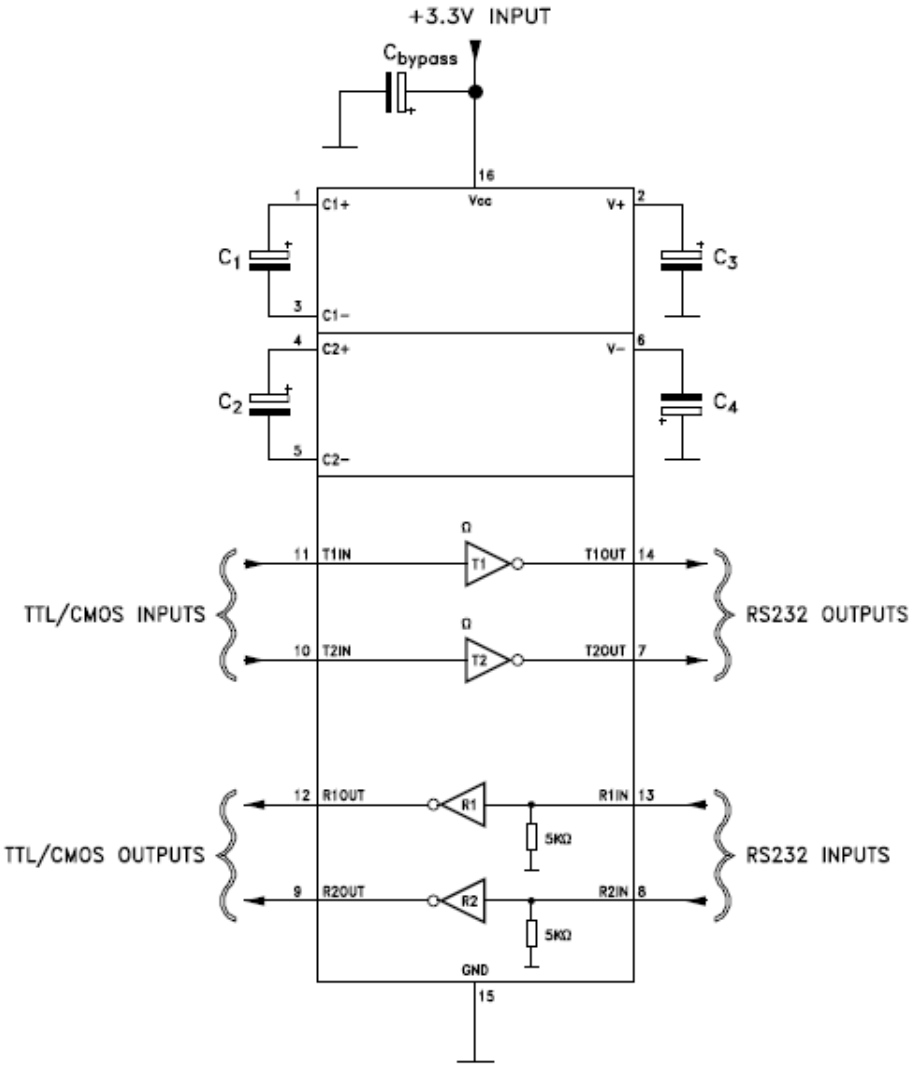
Όπως φαίνεται και από τη διασύνδεση των ακροδεκτών στο σχηματικό της τυπωμένης πλακέτας (βλ. Εικόνα #), η τάση εισόδου, 5V για την υπό κατασκευή εφαρμογή, συνδέεται στις εισόδους των δύο συστημάτων ρύθμισης τάσης διαμέσου πυκνωτών που φιλτράρουν τον υψίσυχνο θόρυβο. Οι είσοδοι ενεργοποίησης είναι σταθερά συνδεδεμένες σε χαμηλό δυναμικό ώστε να παραμένουν διαρκώς σε λειτουργία. Στις εξόδους βρίσκονται πυκνωτές που βελτιώνουν την ευστάθεια των δύο ρυθμιστών. Η έξοδος RESET ελέγχεται από την κατάσταση της εισόδου MR και όταν η δεύτερη βρεθεί σε χαμηλό δυναμικό, η πρώτη παράγει με καθυστέρηση 120 ms ένα αντίστοιχο σήμα που τροφοδοτεί την είσοδο επανεκκίνησης του μικροεπεξεργαστή και ταυτόχρονα προκαλεί το άναμμα ενός LED. Τέλος, οι έξοδοι PG1 και 2 για τους δύο ρυθμιστές τάσης συνδέονται με την είσοδο επανεκκίνησης MR ώστε σε περίπτωση κακής λειτουργίας οποιουδήποτε από τους δύο ρυθμιστές, δηλαδή όταν η αντίστοιχη έξοδος βρίσκεται σε τιμή μικρότερη του 95% της επιθυμητής, να προκαλείται επανεκκίνηση του κυκλώματος.

RS-232 TRANCEIVER IC3G$1

Όπως έχει αναφερθεί, η μετάδοση δεδομένων με χρήση του πρωτοκόλλου RS-232 γίνεται σε επίπεδα τάσης μεγαλύτερα αυτών που μπορούν να ανεχτούν οι ακροδέκτες του μικροεπεξεργαστή. Για τη φυσική διασύνδεση λοιπόν του LPC2129 με τη σειριακή θύρα απαιτείται κάποια ηλεκτρονική διάταξη. Το ρόλο αυτό αναλαμβάνει το ολοκληρωμένο IC3G$1, η λειτουργία του οποίου είναι ακριβώς ίδια με μια σειρά ολοκληρωμένων που θεωρούνται απόγονοι του κλασσικού MAX232. Στα τεχνικά χαρακτηριστικά του περιλαμβάνονται η τροφοδοσία από 3V έως 5V, η μικρή κατανάλωση της τάξης των 300μA και η εγγυμένη ελάχιστη ταχύτητα μετάδοσης δεδομένων σε ρυθμό 250 Kbps. Επιπλέον, οι ακροδέκτες του ολοκληρωμένου από τη μεριά του μικροεπεξεργαστή, δηλαδή η έξοδος του δέκτη και η είσοδος του αποστολέα, λειτουργούν σε επίπεδα τάσης αντίστοιχα της τροφοδοσίας, η οποία είναι στα 3,3V όπως και του μικροεπεξεργαστή, άρα είναι συμβατοί με τους ακροδέκτες του τελευταίου. Από την άλλη, η έξοδος του αποστολέα έχει ικανότητα οδήγησης ±13,2V ενώ η είσοδος μπορεί να ανεχτεί τάσεις από -25 έως +25V. Τα παραπάνω χαρακτηριστικά κάνουν το συγκεκριμένο ολοκληρωμένο συμβατό με τις προδιαγραφές του στάνταρ EIA/TIA-232 και ταυτόχρονα απόλυτα κατάλληλο για χρήση στην ολοκληρωμένη πλακέτα, αφού προσφέρει συμβατότητα με το

μικροεπεξεργαστή, την τροφοδοσία και τους δυνατούς ρυθμούς μεταφοράς δεδομένων. Τέλος, ένα χαρακτηριστικό του ολοκληρωμένου που αξίζει να αναφερθεί εξαιτίας της θέσης του, δηλαδή της άμεσης διασύνδεσής του με τη θύρα DSUB-9, είναι το γεγονός ότι μπορεί να αντέξει εκφόρτιση έως ±15kV στατικού ηλεκτρισμού σύμφωνα με το μοντέλο ανθρώπινου σώματος και έως ±8kV σύμφωνα με το στάνταρ IEC 1000-4-2.

Η χρήση του IC3G$1, όπως φαίνεται στο σχηματικό της ολοκληρωμένης πλακέτας, απαιτεί τέσσερις πυκνωτές των 0,1μF για την εσωτερική αντλία φορτίου όταν η τροφοδοσία είναι στα 3,3V. Οι ακροδέκτες από τη μεριά του μικροεπεξεργαστή συνδέονται απ'ευθείας στους αντίστοιχους αυτού ενώ οι ακροδέκτες από τη μεριά της θύρας συνδέονται διαμέσου προστατευτικών RLC κυκλωμάτων.

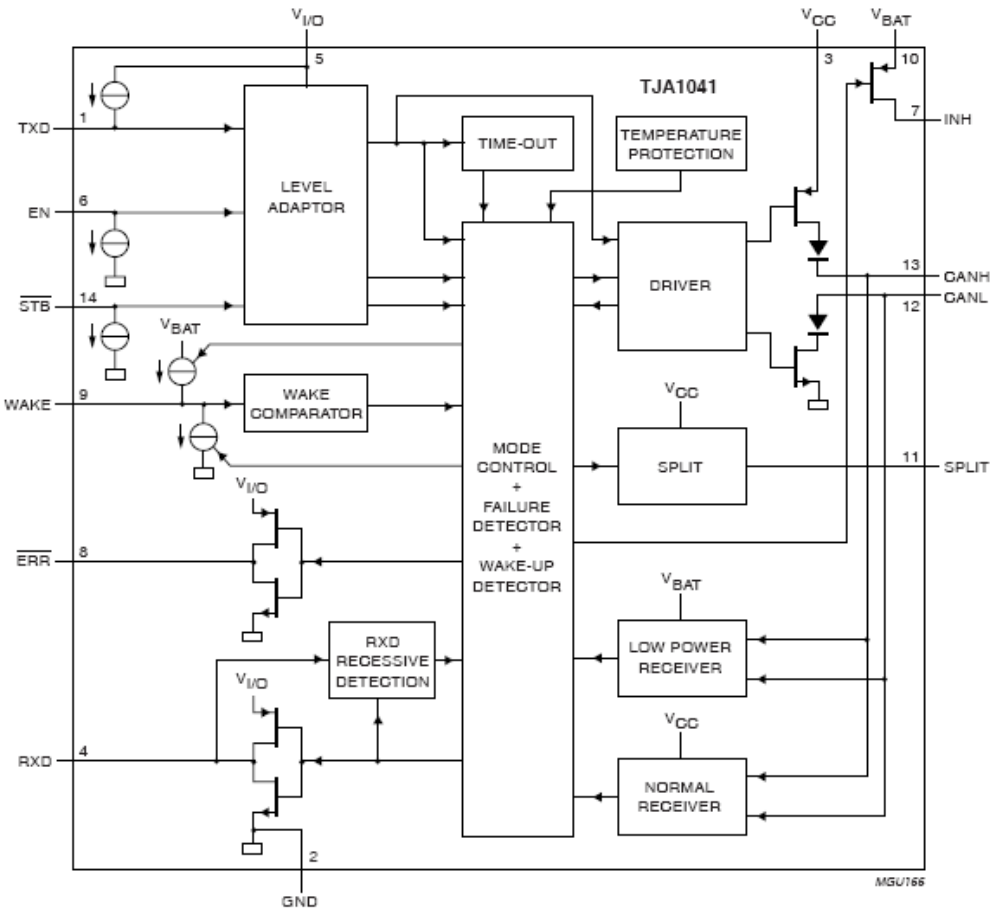


Εικόνα 18: Τυπική διάταξη IC3G$1 [8]

CAN TRANSCEIVER TJA1041

Το ολοκληρωμένο TJA1041 της Philips Semiconductors αποτελεί τη διεπαφή μεταξύ του ελεγκτή του πρωτοκόλλου CAN και το φυσικό δίαυλο του δικτύου CAN σε έναν κόμβο. Είναι συμβατό με το πρότυπο ISO 11898 και προσφέρει δυνατότητα διαφορικής εκπομπής στο δίαυλο και διαφορικής λήψης δεδομένων στον ελεγκτή. Είναι ικανό να λειτουργήσει σε εφαρμογές CAN υψηλής ταχύτητας (έως 1 Mbit/s), έχει μικρή κατανάλωση διαθέτοντας μια σειρά από καταστάσεις μειωμένης λειτουργικότητας και κατανάλωσης ενέργειας, προσαρμόζεται στο επίπεδο τάσης λειτουργίας του ελεγκτή ώστε να είναι συμβατό με διάφορους τύπους ελεγκτών CAN και διαθέτει αρκετές λειτουργίες αναγνώρισης και διάγνωσης σφαλμάτων με δυνατότητα αυτόματης απενεργοποίησης του κόμβου ώστε να μην παρακωλύεται η λειτουργία του διαύλου CAN, άρα και του υπόλοιπου δικτύου, έως ότου αποκατασταθεί η σωστή λειτουργία του κόμβου.

Το μπλοκ διάγραμμα του ολοκληρωμένου είναι το εξής:



Εικόνα 19: Μπλοκ διάγραμμα του TJA1041 [9]

Σε αυτό διακρίνονται τα επιμέρους υποσυστήματα που απαρτίζουν το ολοκληρωμένο και η διασύνδεσή τους τόσο μεταξύ τους όσο και με τους ακροδέκτες.

Οι ακροδέκτες TxD και RxD συνδέονται με τον ελεγκτή CAN για τη μεταφορά των ψηφιακών δεδομένων που λαμβάνονται και αποστέλλονται στο δίαυλο, η σύνδεση με τον οποίο γίνεται στους ακροδέκτες CANH και CANL. Η έξοδος SPLIT συνδέεται (βλ. σχηματικό της header board, Εικόνα #) διαμέσου ενός πυκνωτή με τη γη και διαμέσου δύο αντιστάσεων με τα κανάλια CANH και CANL ώστε να σταθεροποιεί την τάση κοινού σήματος (common mode), στην οποία μπορεί σε συγκεκριμένες μορφές του δικτύου και εξαιτίας κακής λειτουργίας κάποιων κόμβων να παρατηρούνται μεταβατικά φαινόμενα βηματικής μεταβολής τα οποία προκαλούν αυξημένες ηλεκτρομαγνητικές εκπομπές. Ο ακροδέκτης V_{CC} συνδέεται με την τροφοδοσία 5V του κυκλώματος, ο ακροδέκτης $V_{I/O}$ με την τροφοδοσία 3,3V του μικροεπεξεργαστή και ο ακροδέκτης V_{BAT} με την τροφοδοσία από τη μπαταρία, δηλαδή πάλι την κεντρική τροφοδοσία των 5V για τη συγκεκριμένη εφαρμογή. Η σύνδεση του ακροδέκτη $V_{I/O}$ επιτρέπει στο ολοκληρωμένο να ρυθμίζει τα επίπεδα τάσης των TxD και RxD αλλά και των υπολοίπων ακροδεκτών που συνδέονται με τον ελεγκτή ανάλογα με τα επίπεδα τάσης στα οποία λειτουργεί ο ελεγκτής, με αποτέλεσμα να μην απαιτούνται ενδιάμεσα στάδια υλικού. Ο ακροδέκτης WAKE επιτρέπει τον έλεγχο της αφύπνισης του ολοκληρωμένου από κάποια κατάσταση μειωμένης λειτουργικότητας και κατανάλωσης αλλά δεν χρησιμοποιείται στη συγκεκριμένη εφαρμογή, για το λόγο αυτό παραμένει σε υψηλό δυναμικό. Οι ακροδέκτες STB και EN επίσης ελέγχουν τη λειτουργία του transceiver και παράμενουν σε κατάσταση που επιβάλλει κανονική λειτουργία εκτός και αν ρυθμιστούν κατάλληλα οι έξοδοι γενικής χρήσης του επεξεργαστή στις οποίες είναι συνδεδεμένοι. Η έξοδος ERR προσφέρει πληροφορίες για την ύπαρξη προβλημάτων στη διάταξη και είναι διαθέσιμη στην είσοδο/έξοδο γενικής χρήσης P1.19 του LPC2129. Τέλος, ο ακροδέκτης INH προσφέρεται για έλεγχο εξωτερικών συσκευών ανάλογα με την κατάσταση του TJA1041 αλλά δεν χρησιμοποιείται στην εφαρμογή.

Το ολοκληρωμένο παρέχει πολλές λειτουργίες διάγνωσης σφαλμάτων μεταβάλλοντας ανάλογα την κατάσταση ορισμένων εσωτερικών μεταβλητών (flags) και επιβάλλοντας τη μετάβαση σε μη κανονικές λειτουργίες, έτσι ώστε να ελαττώνεται η κατανάλωση ενέργειας και να αποκόβεται ο κόμβος από το δίαυλο όποτε απαιτείται. Με τον τρόπο αυτό, το υπόλοιπο δίκτυο CAN μπορεί να συνεχίσει να λειτουργεί χωρίς τα

προβλήματα που θα δημιουργούσε ο κόμβος αν παρέμενε συνδεδεμένος ενώ δεν λειτουργούσε σωστά. Ταυτόχρονα, στην έξοδο ERR παρέχονται συνοπτικά ορισμένες πληροφορίες για το είδος του προβλήματος.

Τα σφάλματα που αναγνωρίζονται είναι η πτώση της τάσης των ακροδεκτών V_{CC}, $V_{I/O}$ και V_{BAT} για περισσότερο από ένα ορισμένο χρονικό διάστημα, η δυσλειτουργία του διαύλου λόγω βραχυκυκλώματος με κάποια από τις γραμμές V_{CC}, V_{BAT} ή GND, η δυσλειτουργία της σύνδεσης με τον ελεγκτή CAN λόγω προβλήματος του δεύτερου και η υπερθέρμανση. Σε κάθε μία από τις παραπάνω περιπτώσεις γίνονται αυτόματα οι ενέργειες που απαιτούνται για να συνεχίσει η σωστή λειτουργία του διαύλου CAN ή/και να μην καταστραφεί το ολοκληρωμένο και επίσης αυτόματα γίνεται η επιστροφή στην κανονική λειτουργία όταν πάψουν να ισχύουν οι συνθήκες σφάλματος.

ΚΕΝΤΡΙΚΟΣ ΚΟΜΒΟΣ

Λειτουργία

Ο Κεντρικός Κόμβος είναι υπεύθυνος για την επικοινωνία, μέσω της σειριακής θύρας, με το χρήστη, τον έλεγχο της κατάστασης των τριών πλήκτρων κλήσης του ανελκυστήρα από τους αντίστοιχους ορόφους και την ανταλλαγή των κατάλληλων μηνυμάτων, μέσω του διαύλου CAN, με τον Κόμβο Ανελκυστήρα. Για τη σειριακή επικοινωνία χρησιμοποιείται η θύρα DSUB-9 που βρίσκεται ενσωματωμένη στην τυπωμένη πλακέτα (header board). Επομένως, από θέμα κατασκευής, η τυπωμένη πλακέτα αυτή αρκεί να τοποθετηθεί σε διάτρητη πλακέτα κατάλληλης διάταξης ώστε να της παρέχονται η τάση λειτουργίας των 5V, η γείωση και η φυσική διασύνδεση των δύο ακροδεκτών CANH και CANL του πρώτου ελεγκτή CAN. Επιπλέον, για τον καλύτερο έλεγχο της λειτουργίας του κόμβου από το χρήστη, απαιτείται ένας διακόπτης τύπου push-button ο οποίος θα παρέχει το σήμα επανεκκίνησης. Η πίεσή του θα επιτρέπει στο χρήστη να προκαλέσει επανεκκίνηση του μικροεπεξεργαστή.

Η τοποθέτηση της τυπωμένης πλακέτας στη διάτρητη πρέπει να γίνει με τρόπο τέτοιο που να επιτρέπει τη σύνδεση και την αποσύνδεσή της κατά βούληση. Για το λόγο αυτό χρησιμοποιείται η τεχνική που θεωρείται δεδομένη κατά την τοποθέτηση μικρότερων ολοκληρωμένων, δηλαδή η συγκόλληση στη διάτρητη πλακέτα μιας βάσης αντίστοιχης σε αριθμό αποδεκτών με το ολοκληρωμένο. Αυτή η τεχνική επιτρέπει την τοποθέτηση των ολοκληρωμένων χωρίς κόλληση και επομένως την αποσύνδεσή τους, αν αυτό χρειαστεί, ενώ ταυτόχρονα καθιστά δυνατή την κόλληση, με δεδομένο ότι ορισμένα ολοκληρωμένα είναι ευαίσθητα σε ακραίες θερμοκρασίες στους αποδέκτες τους και άρα μπορεί να καταστραφούν από την επαφή τους με το θερμό κολλητήρι. Γίνεται λοιπόν πρώτα η κόλληση της βάσης και το ολοκληρωμένο τοποθετείται σε αυτή μόνο αφού οι ακροδέκτες της βάσης έχουν επανέλθει σε ασφαλή θερμοκρασία. Για την τοποθέτηση χρησιμοποιούνται δύο τυποποιημένες βάσεις των 32 ακροδεκτών, ομαδοποιημένων σε δύο σειρές των 16 ακροδεκτών.

Στο σημείο αυτό πρέπει να σημειωθεί ότι η σωστή σύνδεση των διακοπτών και των πλήκτρων με τους ακροδέκτες του μικροεπεξεργαστή επιβάλλει τη χρήση μιας pull-up αντίστασης. Με τον τρόπο αυτό, όταν ο διακόπτης δεν είναι ενεργοποιημένος, ο ακροδέκτης συνδέεται μέσω της αντίστασης σε υψηλή τάση 5V ενώ όταν ο διακόπτης

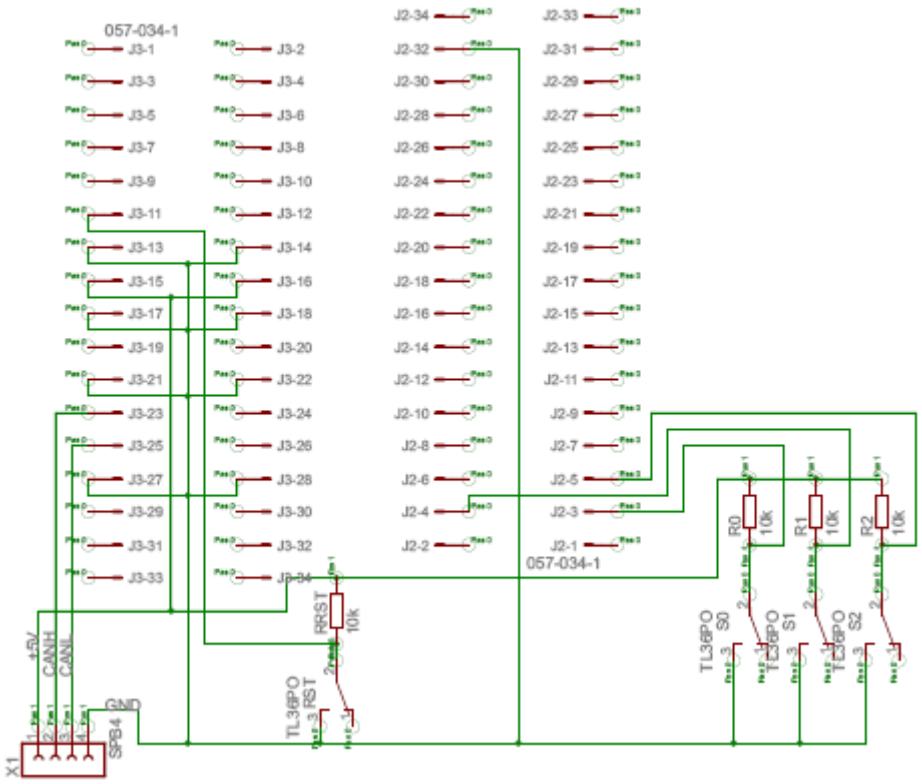
κλείνει το κύκλωμα, ο ακροδέκτης συνδέεται απ'αυθείας με τη γείωση. Η τιμή της αντίστασης πρέπει να είναι τέτοια ώστε η πτώση τάσης κατά μήκος της όταν ο διακόπτης είναι ανοιχτός να φέρνει τον ακροδέκτη σε τάση ικανή να αναγνωριστεί ως υψηλή. Η πτώση τάσης σε αυτήν την περίπτωση οφείλεται στο μικρό ρεύμα διαρροής που τραβάει ο ακροδέκτης του μικροεπεξεργαστή όταν είναι συνδεδεμένος με τα 5V. Το ρεύμα αυτό δίνεται στον πίνακα στατικών χαρακτηριστικών του LPC2129 και έχει τυπική τιμή τα 50μA και μέγιστη τα 150μA. Στον ίδιο πίνακα αναφέρεται ότι το ελάχιστο επίπεδο τάσης που θεωρείται υψηλό είναι τα 2V. Άρα, η μέγιστη επιτρεπτή πτώση τάσης είναι τα 3V και θεωρώντας τη χειρότερη περίπτωση, η μέγιστη τιμή της pull-up αντίστασης είναι

$$R < \frac{3V}{150\mu A} \Rightarrow R < 20k\Omega$$

Η τελική επιλογή της τιμής της αντίστασης γίνεται με βάση την παραπάνω απαίτηση σε συνδυασμό με την ελλάτωση της κατανάλωσης ισχύος στην περίπτωση που ο διακόπτης είναι ενεργοποιημένος. Σε τέτοια κατάσταση, η ισχύς που καταναλώνεται στην αντίσταση και αποτελεί απώλεια είναι αντιστρόφως ανάλογη του τετραγώνου της αντίστασης, επομένως όσο μεγαλύτερη είναι η αντίσταση τόσο μικρότερες είναι οι απώλειες. Για να διατηρηθεί και ένα σημαντικό εύρος ασφαλούς λειτουργίας λαμβάνοντας υπ'όψην τα σφάλματα στις τιμές τόσο των αντιστάσεων όσο και των χαρακτηριστικών του μικροεπεξεργαστή, επιλέχθηκε η τιμή των 10kΩ για όλες τις pull-up αντιστάσεις στους δύο κόμβους.

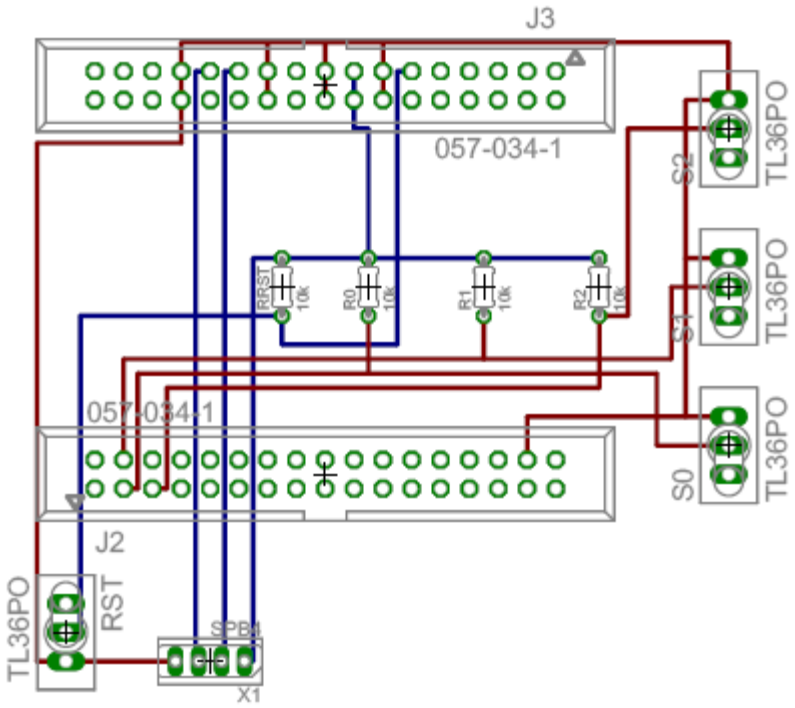
Διάταξη

Το λειτουργικό διάγραμμα της διάτρητης πλακέτας με βάση τη λειτουργία του Κεντρικού Κόμβου που περιγράφηκε παραπάνω καθώς και τα τεχνικά χαρακτηριστικά της τυπωμένης πλακέτας και του LPC2129, οι ακροδέκτες του οποίου σχετίζονται με τους ακροδέκτες της πρώτης, δίνεται παρακάτω.

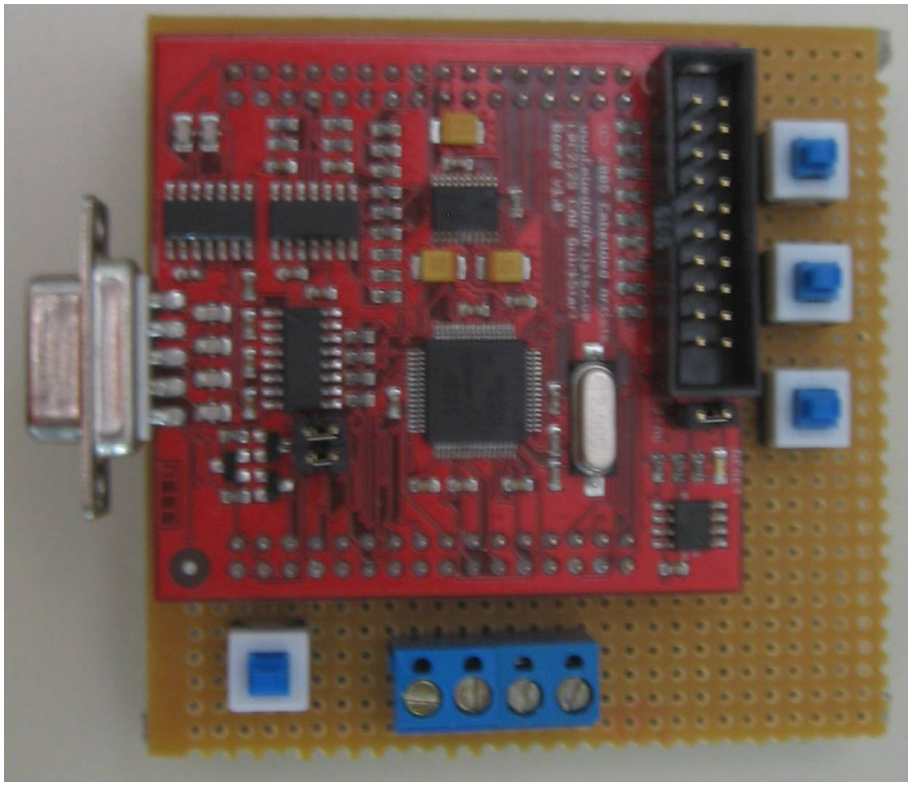


Εικόνα 20: Λειτουργικό διάγραμμα του Κεντρικού Κόμβου

Το σχέδιο των συνδέσεων στη διάτρητη πλακέτα, όπως θα φαίνονταν από το επάνω μέρος, δίνεται στην επόμενη εικόνα. Η κατασκευή αυτή καθεαυτή γίνεται στο κάτω μέρος ακολουθώντας το ίδιο σχέδιο αντεστραμμένο κατά τον κατακόρυφο άξονα.



Εικόνα 21: Σχέδιο κατασκευής της διάτρητης πλακέτας του Κεντρικού Κόμβου



Εικόνα 22: Ο Κεντρικός Κόμβος

ΚΟΜΒΟΣ ΑΝΕΛΚΥΣΤΗΡΑ

Λειτουργία

Ο Κόμβος Ανελκυστήρα είναι υπεύθυνος για τον έλεγχο της λειτουργίας του ανελκυστήρα. Συγκεκριμένα, καθορίζει την ένδειξη του 7-segment display, τα τμήματα του οποίου είναι συνδεδεμένα στους ακροδέκτες P0.4 έως P0.10, ελέγχει τα πλήκτρα ορόφων που είναι συνδεδεμένα στους ακροδέκτες P0.17, P0.18 και P0.19 για το ισόγειο, τον πρώτο και το δεύτερο όροφο αντίστοιχα, τα πλήκτρα STOP και ALARM που συνδέονται στους ακροδέκτες P0.20 και P0.21, την κατάσταση του διακόπτη που επιτελεί το ρόλο αισθητήρα βάρους και συνδέεται στον ακροδέκτη P0.16 και την κατάσταση του μαγνητικού αισθητήρα που είναι συνδεδεμένος στον ακροδέκτη P0.15. Πέραν των παραπάνω, ρυθμίζει τη φορά περιστροφής του κινητήρα καθορίζοντας το επίπεδο τάσης στον ακροδέκτη P0.2 και διαμορφώνει σήματα παλμών για την περιστροφή του βηματικού κινητήρα στον ακροδέκτη P0.3. Τέλος, φροντίζει για την επικοινωνία με τον Κεντρικό Κόμβο διαμέσου του διαύλου CAN, με τον οποίο ανταλλάζει μηνύματα για την τρέχουσα κατάσταση του ανελκυστήρα καθώς και τις εντολές του χρήστη για τη μετακίνησή του.

Για την υλοποίηση αυτών των λειτουργιών είναι απαραίτητες οι αντίστοιχες φυσικές συνδέσεις των ακροδεκτών της τυπωμένης πλακέτας. Οι ακροδέκτες εξόδου συνδέονται στις εισόδους δύο ολοκληρωμένων που περιλαμβάνουν από 6 αντιστροφείς, οι έξοδοι των οποίων είναι με τη σειρά τους συνδεδεμένες με τους ακροδέκτες του 7-segment display. Το display που χρησιμοποιήθηκε είναι κοινής ανόδου και για τη σύδεσή του με την πηγή των 5V παρεμβάλλονται σύο αντιστάσεις σε σειρά συνολικής τιμής 200Ω. Η αντίσταση αυτή καθορίζει το ρεύμα λειτουργίας του display σε περίπου $\frac{3V}{200\Omega} = 15mA$, με δεδομένο ότι η πτώση τάσης κατά μήκος του display δίνεται στο φύλλο δεδομένων του με τυπική τιμή 2V. Η ένταση αυτή, που απαιτείται για τη φωτοβολία των LEDs, παρέχεται από την τροφοδοσία και καταλήγει στους ακροδέκτες του ολοκληρωμένου 74LS04 που περιλαμβάνει τους αντιστροφείς και είναι ικανό να διαχειριστεί ρεύμα τέτοιας έντασης. Οι ακροδέκτες εισόδου συνδέονται στους διακόπτες και στην τάση των 5V μέσω pull-up αντίστασης, όπως περιγράφηκε παραπάνω. Ειδικά για το μαγνητικό αισθητήρα, μεταξύ του ακροδέκτη του μικροεπεξεργαστή και της pull-

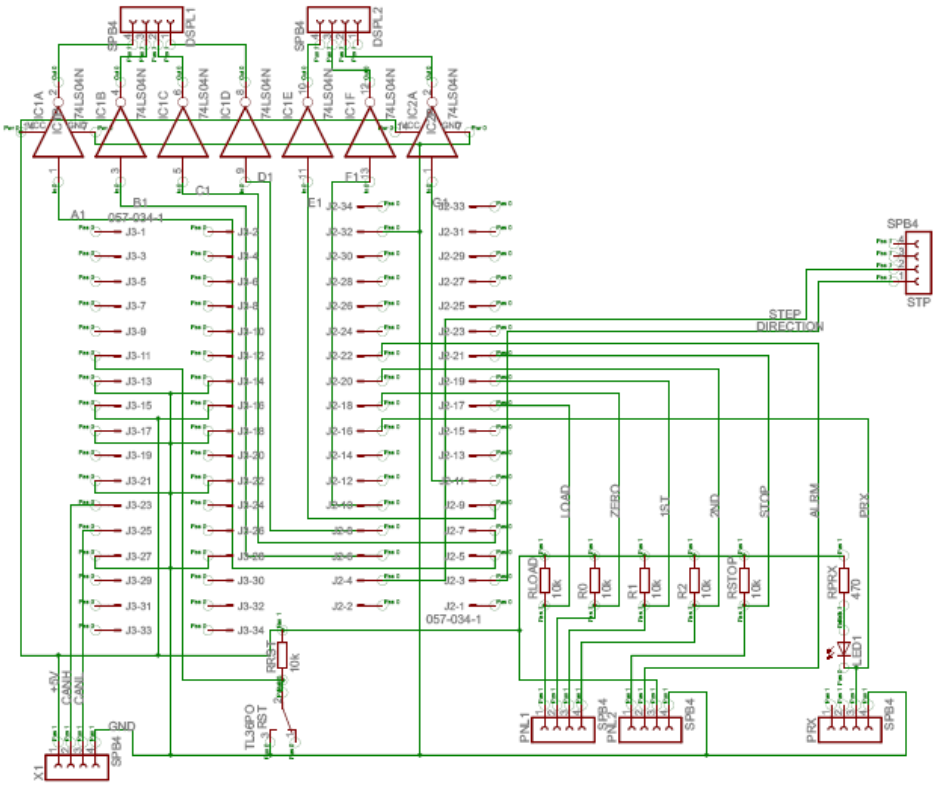
up αντίστασης παρεμβάλλεται LED, το οποίο ανάβει όταν ο αισθητήρας βρίσκεται κοντά σε έναν από τους μόνιμους μανήτες που έχουν τοποθετηθεί στους ορόφους της μακέτας.

Όπως και στον Κεντρικό Κόμβο, η επανεκκίνηση του μικροεπεξεργαστή του Κόμβου Ανελκυστήρα ελέγχεται από ένα διακόπτη τύπου push-button. Η τροφοδοσία, η γείωση και τα δύο σήματα CANH και CANL παρέχονται στη διάτρητη πλακέτα ενώ αντίστοιχα παρέχονται τροφοδοσία, γείωση και τα σήματα ελέγχου που θα τροφοδοτήσουν το driver του βηματικού κινητήρα. Τέλος, τόσο τα σήματα εισόδου από τους διακόπτες, τα πλήκτρα και τον αισθητήρα όσο και τα σήματα εξόδου για το 7-segment display παρέχονται στον μικροεπεξεργαστή μέσω ακροδεκτών πλακέτας, καθώς η φυσική τοποθέτησή τους είναι στο εξωτερικό τοίχωμα του ανελκυστήρα. Η τοποθέτηση της τυπωμένης πλακέτας γίνεται με τον ίδιο τρόπο που χρησιμοποιήθηκε και για τον Κεντρικό Κόμβο.

Ο Κόμβος Ανελκυστήτα είναι τοποθετημένος στο εσωτερικό μιας μινιατούρας ανελκυστήρα μήκους 12cm, πλάτους 10cm και ύψους 14cm. Αντίστοιχο είναι το ύψος των ορόφων της μακέτας και οι διαστάσεις του φρεατίου που είναι κατασκευασμένο σε αυτήν. Η μπροστινή πλευρά του ανελκυστήρα περιλαμβάνει τα πλήκτρα και το 7-segment display καθώς και το LED που είναι αναμμένο όσο ο διακόπτης ALARM είναι ενεργοποιημένος. Η διάτρητη πλακέτα του κόμβου βρίσκεται στο κάτω μέρος και στο πλάι της είναι ο μαγνητικός αισθητήρας, με τρόπο τέτοιο ώστε να ευθυγραμμίζεται με τους μόνιμους μαγνήτες που είναι τοποθετημένοι επάνω στην πλάκα καθενός από τους ορόφους της μακέτας όταν ο ανελκυστήρας βρίσκεται ακριβώς στο επίπεδο του ορόφου.

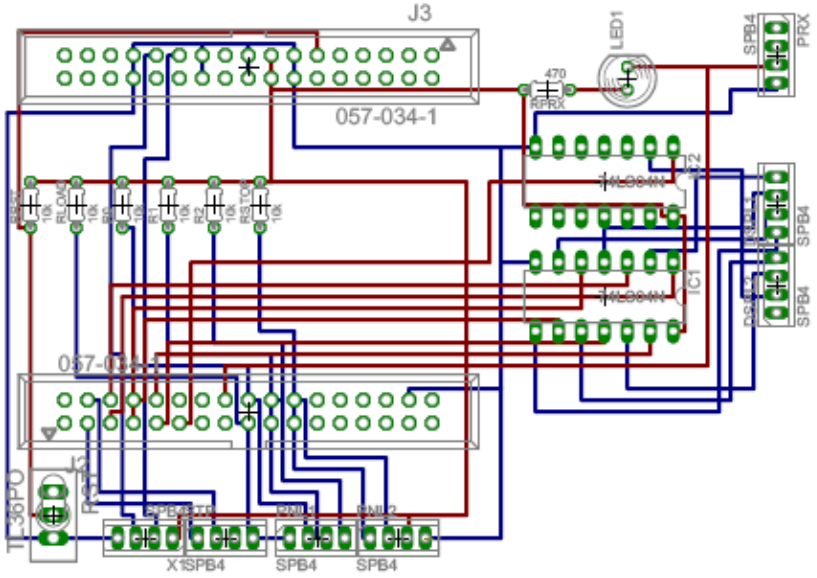
Διάταξη

Ανάλογα με τη διάταξη του Κεντρικού Κόμβου, στη συνέχεια δίνονται τα σχέδια του Κόμβου Ανελκυστήρα.

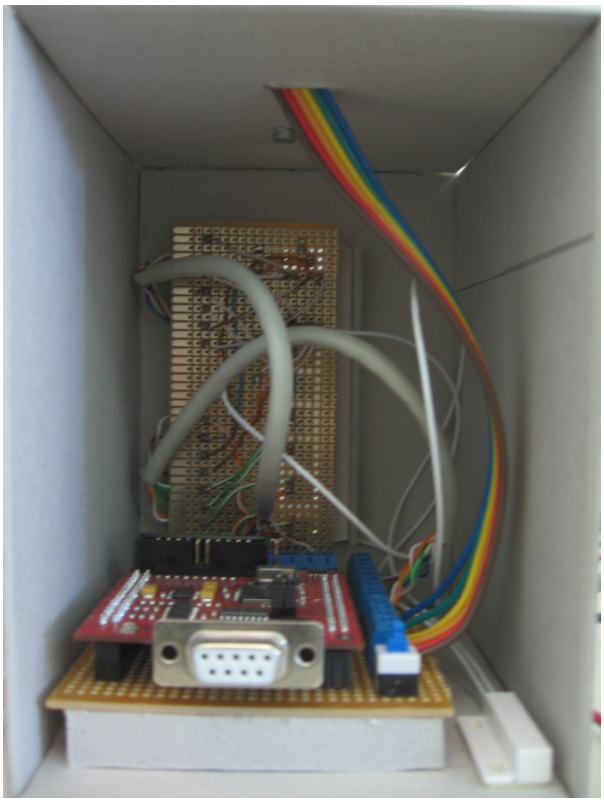


Εικόνα 23: Λειτουργικό διάγραμμα του Κόμβου Ανελκυστήρα

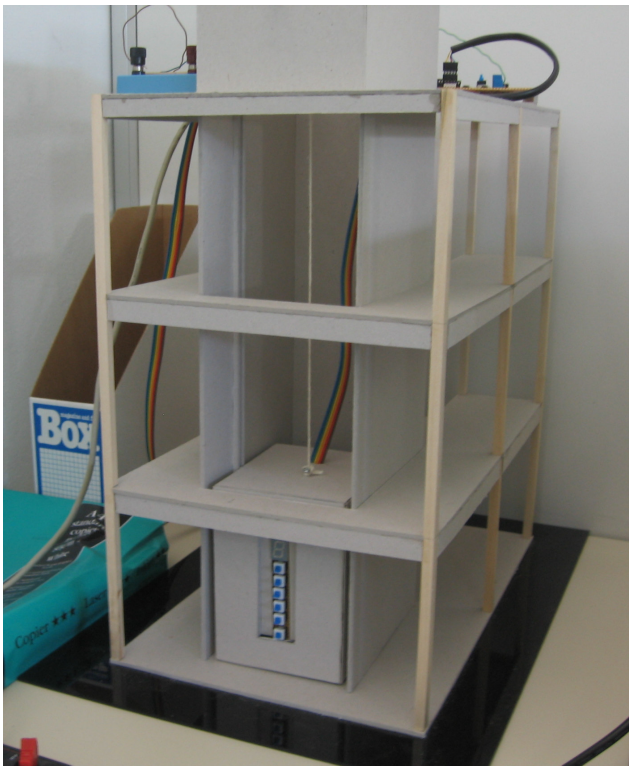
Το σχέδιο των συνδέσεων στη διάτρητη πλακέτα, όπως θα φαίνονταν από το επάνω μέρος, δίνεται στην επόμενη εικόνα. Η κατασκευή αυτή καθεαυτή γίνεται στο κάτω μέρος ακολουθώντας το ίδιο σχέδιο αντεστραμμένο κατά τον κατακόρυφο άξονα.



Εικόνα 24: Σχέδιο κατασκευής της διάτρητης πλακέτας του Κόμβου Ανελκυστήρα



Εικόνα 25: Ο Κόμβος Ανελκυστήρα



Εικόνα 26: Η μακέτα

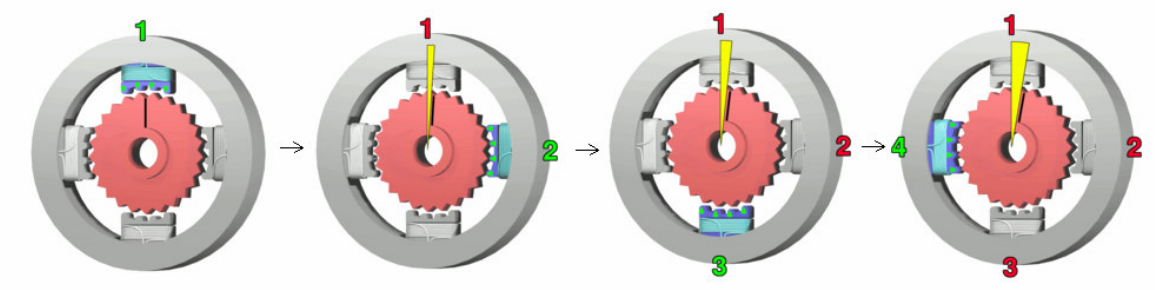
ΒΗΜΑΤΙΚΟΣ ΚΙΝΗΤΗΡΑΣ

Αρχή λειτουργίας

Το θεωρητικό μοντέλο ενός ηλεκτρικού βηματικού κινητήρα είναι ένας σύγχρονος κινητήρας AC με αυξημένο πλήθος πόλων τόσο στο ρότορα όσο και στο στάτορα, με δεδομένο ότι δεν έχουν κανένα κοινό παρονομαστή. Η κατασκευή ενός τέτοιου κινητήρα γίνεται χρησιμοποιώντας μαλακό μαγνητικό υλικό και για το ρότορα και για το στάτορα, στο οποίο δίνεται εξωτερικά οδοντωτό σχήμα. Οι σύγχρονοι βηματικοί κινητήρες, όπως αυτός που χρησιμοποιείται στην εφαρμογή, είναι υβριδικοί ως προς την κατασκευή τους, δηλαδή αποτελούνται από συνδυασμό μαλακού μαγνητικού υλικού και μόνιμων μαγνητών.

Η περιστροφή του κινητήρα, σε αντίθεση με τους συνηθισμένους, προκαλείται όχι από τη διαρροή κάποιου τυλίγματος από ρεύμα αλλά από την κατάλληλη ακολουθία ρευμάτων στα τυλίγματα. Συγκεκριμένα, το σταθερό μέρος του κινητήρα μπορεί να θεωρηθεί ότι αποτελείται από μια σειρά οδοντωτών ηλεκτρομαγνητών που βρίσκονται γύρω από ένα κεντρικό οδοντωτό μεταλλικό ρότορα. Οι σταθεροί ηλεκτρομαγνήτες διαθέτουν ξεχωριστά τυλίγματα ώστε να ενεργοποιούνται αυτόνομα. Όταν ο πρώτος ηλεκτρομαγνήτης διαρρέεται από ηλεκτρικό ρεύμα, τα «δόντια» του ρότορα έλκονται και ευθυγραμμίζονται με αυτά του ενεργού ηλεκτρομαγνήτη. Στη συνέχεια απενεργοποιείται ο πρώτος και ενεργοποιείται ο δεύτερος ηλεκτρομαγνήτης, τα «δόντια» του οποίου είναι εκ κατασκευής ελαφρώς μετατοπισμένα από αυτά του ρότορα, όπως βρίσκονται στην πρώτη θέση. Η ενεργοποίηση μόνο του δεύτερου ηλεκτρομαγνήτη προκαλεί μια μικρή περιστροφή στο ρότορα ώστε να ευθυγραμμιστεί εκ νέου με το δεύτερο ηλεκτρομαγνήτη. Η διαδικασία συνεχίζεται ενεργοποιώντας διαδοχικά όλους τους σταθερούς ηλεκτρομαγνήτες και προκαλώντας σε κάθε μετάβαση την περιστροφή του κινητήρα κατά τις μοίρες που καθορίζονται από τη μετατόπιση που έχει δώσει ο κατασκευαστής στα δόντια των ηλεκτρομαγνητών και του ρότορα.

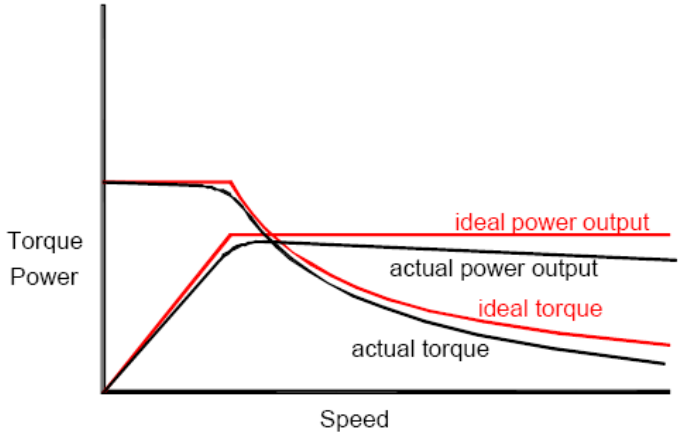
Η διαδικασία περιγράφεται στις παρακάτω εικόνες, με παράδειγμα ένα βηματικό κινητήρα τεσσάρων φάσεων, με τέσερις δηλαδή ανεξάρτητους ηλεκτρομαγνήτες.



Εικόνα 27: Περιστροφή ενός βηματικού κινητήρα [10]

Ως μετατροπέας ενέργειας, ο βηματικός κινητήρας παρέχει θεωρητικά σταθερή ισχύ εξόδου. Η ροπή του είναι ανάλογη του ρεύματος που διαρρέει τους ηλεκτρομαγνήτες του και, εφόσον η ισχύς του είναι σταθερή, το γινόμενο ταχύτητας περιστροφής και ροπής είναι σταθερό. Επομένως, η ροπή του κινητήρα αυτού είναι αντιστρόφως ανάλογη της ταχύτητας περιστροφής. Επειδή κάτι τέτοιο θα σήμαινε πολύ μεγάλες εντάσεις ρεύματος σε χαμηλές ταχύτητες, το κύκλωμα οδήγησης φροντίζει να κόβει την ένταση του ρεύματος όταν, μειούμενης της ταχύτητας περιστροφής, αυτό αυξάνεται πάνω από ένα ορισμένο όριο. Με τον τρόπο αυτό, στις ταχύτητες όπου ισχύει το ψαλίδισμα του ρεύματος, η ισχύς εξόδου παύει να είναι σταθερή αλλά διαμορφώνεται ανάλογη της ταχύτητας περιστροφής.

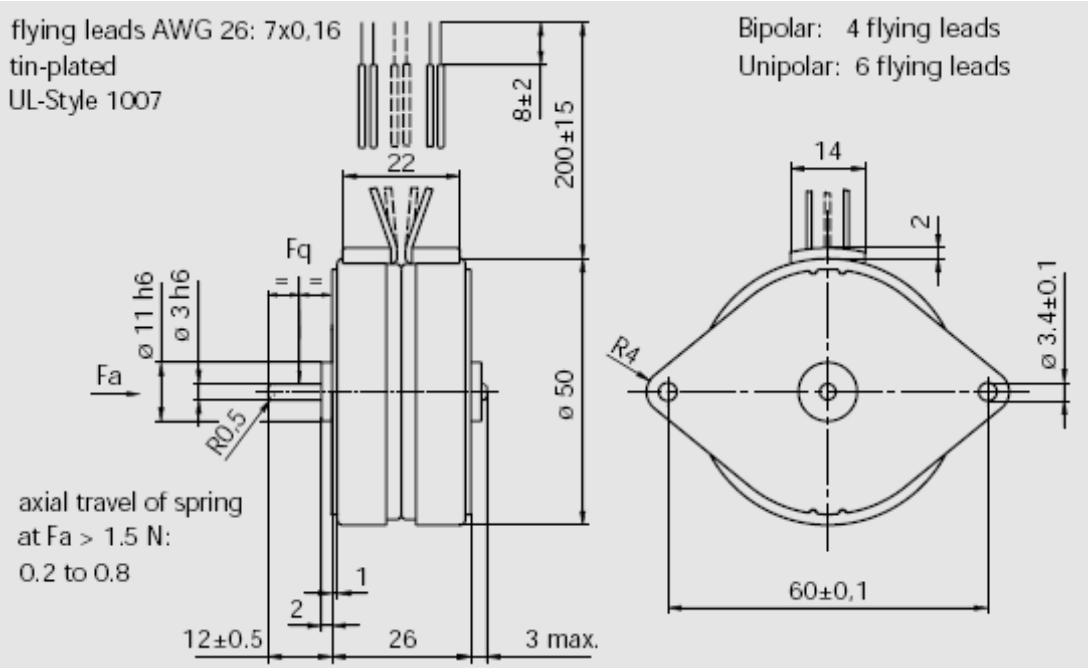
Οι διαφορές μεταξύ του θεωρητικού μοντέλου και ενός πραγματικού βηματικού κινητήρα είναι ότι ο δεύτερος έχει μη μηδενική ωμική αντίσταση τυλιγμάτων ενώ οι μαγνήτες του υφίστανται μαγνητικό κορεσμό και απώλειες υστέρησης. Το αποτέλεσμα είναι η ισχύς εξόδου να ελαττώνεται από ένα σημείο και μετά όσο αυξάνεται η ταχύτητα, καθώς αυξάνεται η επίδραση της παραμένουσας ροπής. Το φαινόμενο αυτό περιγράφεται στο παρακάτω γράφημα.



Εικόνα 28: Θεωρητική και πρακτική σχέση ισχύος-ταχύτητας περιστροφής [11]

Ο βηματικός κινητήρας που χρησιμοποιείται στην εφαρμογή είναι ο RDM 57/6 της εταιρείας Berger Lahr. Είναι διφασικός, υβριδικός και διπολικός, δηλαδή έχει ένα τύλιγμα για κάθε φάση και άρα τέσσερις ακροδέκτες. Υπάρχουν βηματικοί κινητήρες με δύο τυλίγματα ανά φάση, ώστε το ένα τύλιγμα να αντιστοιχεί στη μία και το άλλο στην αντίθετη φορά ρεύματος. Στους διπολικούς όμως η αντιστροφή του ρεύματος στο τύλιγμα κάθε φάσης είναι ευθύνη του αντίστοιχου κυκλώματος οδήγησης. Οι διπολικοί βηματικοί κινητήρες προσφέρουν μεγαλύτερη ισχύ στο ίδιο βάρος εξαιτίας της καλύτερης χρήσης των τυλιγμάτων αλλά απαιτούν πιο πολύπλοκο κύκλωμα οδήγησης.

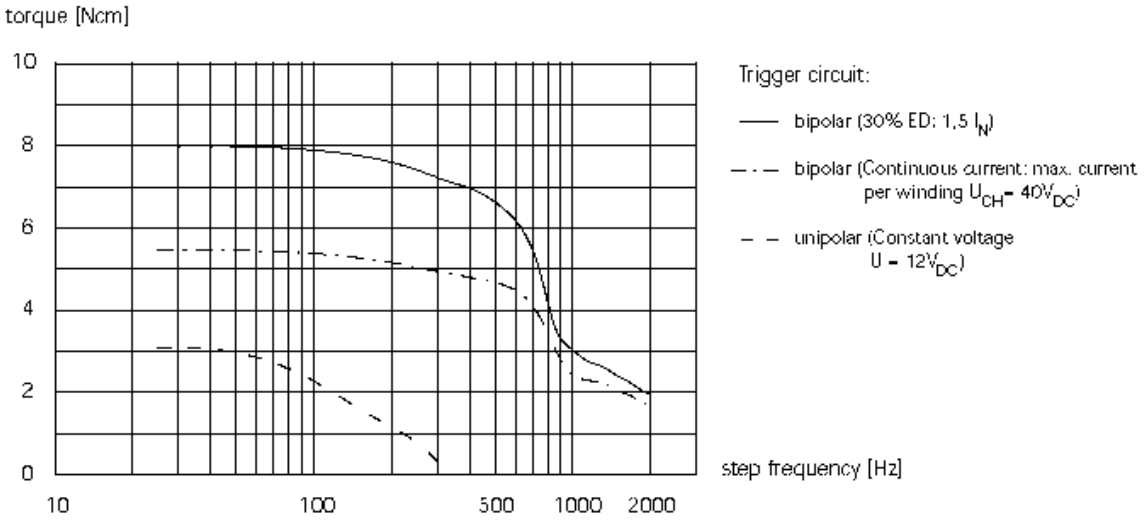
Το μηχανολογικό σχέδιο του κινητήρα που χρησιμοποιείται είναι το παρακάτω:



Εικόνα 29: Μηχανολογικό σχέδιο του βηματικού κινητήρα RDM 57/6 [12]

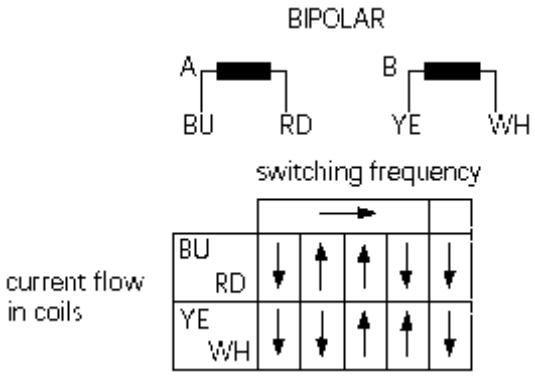
Σύμφωνα με τα τεχνικά χαρακτηριστικά που δίνονται από τον κατασκευαστή, το βήμα του κινητήρα είναι 15°, η αντίσταση κάθε τυλίγματος είναι 15Ω και η ονομαστική ροπή του είναι 5,5Ncm που επιτυγχάνεται με ονομαστική ένταση 0,4A ανά τύλιγμα, που αντιστοιχεί σε ονομαστική τάση 6V. Συνήθως, η τάση των βηματικών κινητήρων μπορεί να φτάσει πολλαπλάσια της ονομαστικής ώστε να επιτευχθεί μεγαλύτερη ροπή. Στην εφαρμογή, ο driver τροφοδοτεί τον κινητήρα με τάση 10V.

Η χαρακτηριστική καμπύλη είναι η εξής:



Εικόνα 30: Χαρακτηριστική καμπύλη του βηματικού κινητήρα RDM 57/6 [12]

Για την περιστροφή του κινητήρα, ο κατασκευαστής δίνει το παρακάτω διάγραμμα με την ακολουθία των ρευμάτων που πρέπει να διαρρέουν τα δύο τυλίγματα του κινητήρα.



Πίνακας 42: Ακολουθία ρευμάτων για την περιστροφή του βηματικού κινητήρα RDM 57/6 [12]

Η ακολουθία αυτή επιτυγχάνεται με κατάλληλη διάταξη του κυκλώματος οδήγησης που θα αναλυθεί στη συνέχεια ως driver.

Εφαρμογές

Η κύρια εφαρμογή των βηματικών κινητήρων είναι σε εργασίες όπου απαιτείται έλεγχος θέσης. Το πλεονέκτημά τους απέναντι στους σερβοκινητήρες που επίσης συνηθίζονται σε

τέτοιες εφαρμογές είναι η δυνατότητά τους να προσφέρουν καλή ακρίβεια και μικρά σφάλματα χωρίς τη χρήση ανάδρασης. Επομένως, έχουν μικρότερες απαιτήσεις στο κύκλωμα ελέγχου και μεγαλύτερη αξιοπιστία. Όπου απαιτείται ακρίβεια σε επίπεδα αντίστοιχα με αυτά που επιτυγχάνονται από κυκλώματα σερβοκινητήρων, υπάρχουν συστήματα ανάδρασης θέσης που τυπικά περιλαμβάνουν οπτικούς κωδικοποιητές και χρησιμοποιούνται αποτελεσματικά με βηματικούς κινητήρες. Επιπλέον, η χρήση πολυπλοκότερων κυκλωμάτων οδήγησης επιτρέπει την αύξηση της ανάλυσης της κίνησης ενός βηματικού κινητήρα, χωρίζοντας ένα κανονικό βήμα θεωρητικά σε έως 256 μικρο-βήματα, παρόλο που στην πράξη οι φυσικοί περιορισμοί του κινητήρα δεν επιτρέπουν την επίτευξη της ακρίβειας αυτής.

Στο χώρο της βιομηχανίας, μεγάλοι βηματικοί κινητήρες χρησιμοποιούνται σε συστήματα επιλογής και τοποθέτησης υψηλής ταχύτητας, σε μηχανές CNC, σε μηχανές συσκευασίας, σε βαλβίδες ελέγχου ροής κ.α. Σε οπτικά συστήματα, μικρότεροι βηματικοί κινητήρες οδηγούν συστήματα γραμμικής μετακίνησης, γωνιόμετρα και περιστρεφόμενους καθρέπτες. Σε αυτόματους χημικούς αναλυτές, βηματικοί κινητήρες σε συνδεσμολογία ανοικτού βρόχου αναλαμβάνουν την κατάλληλη τοποθέτηση των φιαλιδίων με τα δείγματα και τον έλεγχο αυτόματων πιπεττών. Τέλος, βηματικοί κινητήρες μικρότερων διαστάσεων είναι ευρέως διαδεδομένοι σε οδηγούς δισκέτας, σε σαρωτές, σε εκτυπωτές σε plotter και άλλες συσκευές.

DRIVER ΤΟΥ ΒΗΜΑΤΙΚΟΥ ΚΙΝΗΤΗΡΑ

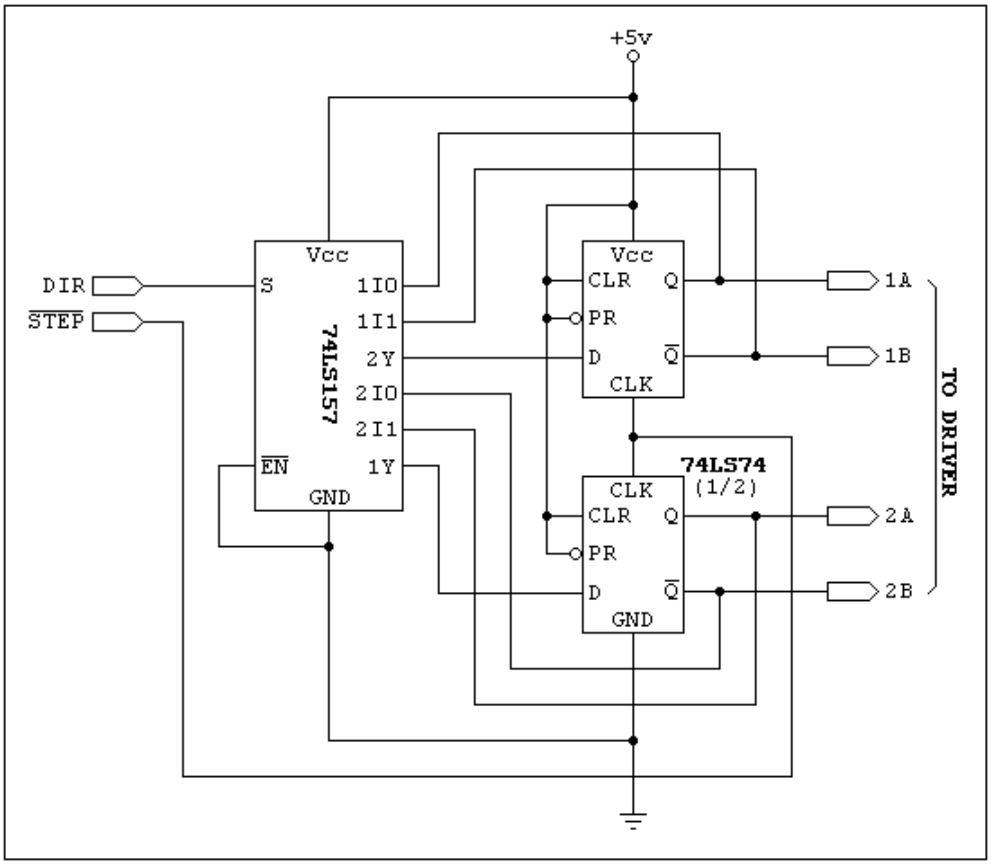
Λειτουργία

Όπως αναλύθηκε προηγουμένως, ο κινητήρας απαιτεί μια συγκεκριμένη ακολουθία από εναλλαγές ρευμάτων στα τυλίγματά του. Επιπλέον, τα ρεύματα αυτά πρέπει να είναι σχετικά μεγάλης έντασης. Επειδή οι ακροδέκτες του μικροεπεξεργαστή που βρίσκεται στον Κόμβο 2 δεν μπορούν να οδηγήσουν ρεύματα τέτοιας έντασης, απαιτείται κάποιο ενδιάμεσο κύκλωμα που θα επικοινωνεί με τον Κόμβο 2 και θα οδηγεί κατάλληλα το βηματικό κινητήρα σύμφωνα με τα σήματα που λαμβάνει από τον κόμβο.

Ο driver του κινητήρα αποτελείται από δύο μέρη:

- Ένα λογικό που αναλαμβάνει την ανάγνωση των δύο σημάτων ελέγχου του βηματικού κινητήρα, δηλαδή ένα σήμα επιπέδου τάσης που ορίζει τη φορά περιστροφής (DIRECTION) και ένα σήμα παλμού που δίνει την εντολή περιστροφής κατά ένα βήμα (STEP). Το υπο-κύκλωμα αυτό μετατρέπει την πληροφορία εισόδου που δίνεται στα δύο αυτά σήματα στα σήματα τάσης που τροφοδοτούν τα τυλίγματα του κινητήρα.
- Ένα υπο-κύκλωμα που απλά λαμβάνει τα σήματα τάσης που παράγονται από το προηγούμενο και τροφοδοτεί ακριβώς ανάλογα τους πραγματικούς ακροδέκτες του βηματικού κινητήρα με την απαιτούμενη μεγαλύτερη τάση από αυτή στην οποία λειτουργεί το λογικό κύκλωμα και το αντίστοιχο ρεύμα.

Το λογικό διάγραμμα του πρώτου υπο-κυκλώματος είναι το εξής:



Εικόνα 31: Λογικό διάγραμμα του driver του βηματικού κινητήρα

Η έξοδος του κυκλώματος εξαρτάται από τα δύο σήματα εισόδου. Όταν το σήμα DIRECTION είναι μηδέν, κάθε παλμός που δίνεται στο σήμα STEP ως είσοδος στα δύο D flip-flop προκαλεί τις μεταβάσεις που περιγράφονται στον αριστερό πίνακα που ακολουθεί, με δεδομένο ότι στην περίπτωση αυτή οι έξοδοι του πολυπλέκτη ακολουθούν τις πρώτες εισόδους, δηλαδή D2=1Y=1I0 και D1=2Y=2I0. Στην αντίθετη περίπτωση, που το σήμα DIRECTION είναι μονάδα, ισχύουν οι σχέσεις D2=1Y=1I1 και D1=2Y=2I1 και οι μεταβάσεις δίνονται στον πίνακα δεξιά.

	t_1	t_2	t_3	t_4
Q_1	0	1	1	0
Q_1’	1	0	0	1
Q_2	0	0	1	1
Q_2’	1	1	0	0

	t_1	t_2	t_3	t_4
Q_1	0	0	1	1
Q_1’	1	1	0	0
Q_2	0	1	1	0
Q_2’	1	0	0	1

Πίνακας 43: Πίνακας μεταβάσεων του driver

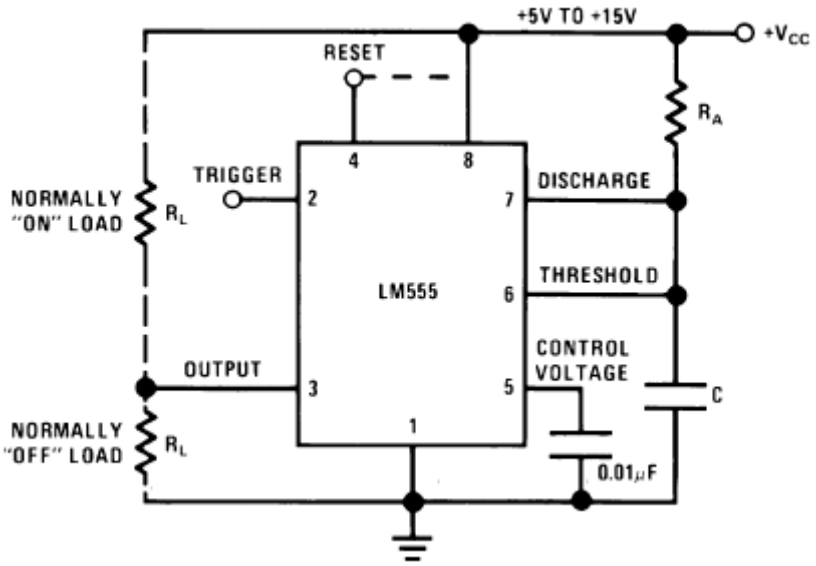
Αντιστοιχίζοντας την έξοδο Q_1 με τον μπλε ακροδέκτη του βηματικού κινητήρα, την Q_1' με τον κόκκινο, την Q_2 με τον κίτρινο και την Q_2' με τον άσπρο και θεωρώντας ότι το δεύτερο υπο-κύκλωμα του driver μετατρέπει τα λογικά αυτά σήματα σε τάσεις με θετική λογική (0 = 0V, 1 = 10V) παρατηρείται ότι η πρώτη ακολουθία προκαλεί την ωρολογιακή περιστροφή του βηματικού κινητήρα καθώς συμπίπτει με τον πίνακα που δίνεται από τον κατασκευαστή ενώ η δεύτερη είναι η ακριβώς αντίθετη και άρα προκαλεί την περιστροφή κατά την αντίθετη φορά. Ο πολυπλέκτης βρίσκεται στο ολοκληρωμένο 74LS157 ενώ τα δύο flip-flop στο ολοκληρωμένο 74LS74.

Το δεύτερο υπο-κύκλωμα του driver αποτελείται από ένα ολοκληρωμένο, το L293B και 8 προστατευτικές διόδους. Το ολοκληρωμένο είναι ένα κύκλωμα τεσσάρων push-pull διατάξεων ικανών να διοχετεύσουν ρεύμα έως 1A σε κάθε έξοδο. Η τάση κάθε εξόδου ακολουθεί την τάση της αντίστοιχης εισόδου. Το επίπεδο της τάσης εξόδου καθορίζεται από την τάση που τροφοδοτεί τον ακροδέκτη Vs ενώ οι είσοδοι απαιτούν την τυπική τάση τροφοδοσίας των κυκλωμάτων TTL στα 5V. Οι 8 δίοδοι λειτουργούν ως προστασία από τα ρεύματα που προκαλούνται σε περίπτωση άσκησης εξωτερικής αντίστροφης ροπής στον κινητήρα.

Ένα χαρακτηριστικό που προστέθηκε στον driver είναι η επιλογή μεταξύ αυτόματης και χειροκίνητης λειτουργίας που γίνεται από ένα διπλό διακόπτη δύο θέσεων. Όταν επιλέγεται η αυτόματη λειτουργία, τα σήματα STEP και DIRECTION λαμβάνονται από τις εξωτερικές εισόδους της διάτρητης πλακέτας που, στη συγκεκριμένη εφαρμογή, παράγονται από τον Κόμβο 2. Όταν ο driver βρίσκεται σε χειροκίνητη λειτουργία, το σήμα DIRECTION παράγεται από ένα διακόπτη δύο θέσεων που συνδέει το σταθερό του ακροδέκτη είτε με την τάση τροφοδοσίας είτε με τη γη και το σήμα STEP από ένα ολοκληρωμένο 555 σε συνδεσμολογία one-shot, προκαλώντας έναν παλμό όταν του δοθεί η αντίστοιχη εντολή μέσω ενός push button που βρίσκεται σταθερά συνδεδεμένο στην τάση τροφοδοσίας και όταν πιέζεται συνδέεται με τη γη.

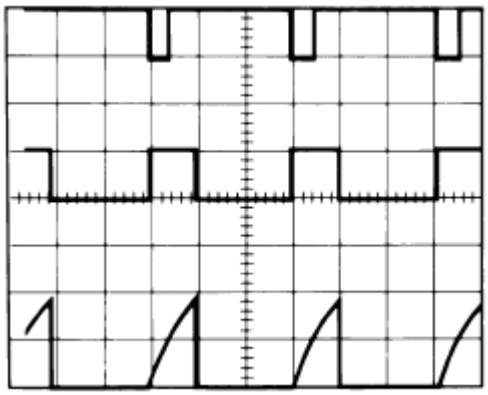
Η προσθήκη του 555 είναι αναγκαία διότι το πάτημα του push button εκ κατασκευής παράγει μία ακανόνιστη ακολουθία παλμών. Το φαινόμενο αυτό οφείλεται στην ταλάντωση των κινούμενων μερών στο εσωτερικό του διακόπτη που έχουν ως αποτέλεσμα τη μη σταθερή ηλεκτρική ένωση των ακροδεκτών του για το χρονικό διάστημα που απαιτείται μέχρι την παύση της ταλάντωσης αυτής. Η αντιμετώπιση του προβλήματος, με δεδομένο ότι η είσοδος ρολογιού των flip-flop που δέχεται τους

παλμούς είναι εξαιρετικά ευαίσθητη, πρέπει να γίνει με κύκλωμα one-shot, δηλαδή κύκλωμα ικανό να παράγει έναν και μόνο «καθαρό» παλμό μόλις λαμβάνει στην είσοδό του τουλάχιστον έναν παλμό, αγνοώντας τυχόν επόμενους. Η λειτουργία αυτή υλοποιείται με το γνωστό ολοκληρωμένο 555 σε κατάλληλη διάταξη που φαίνεται στην επόμενη εικόνα.



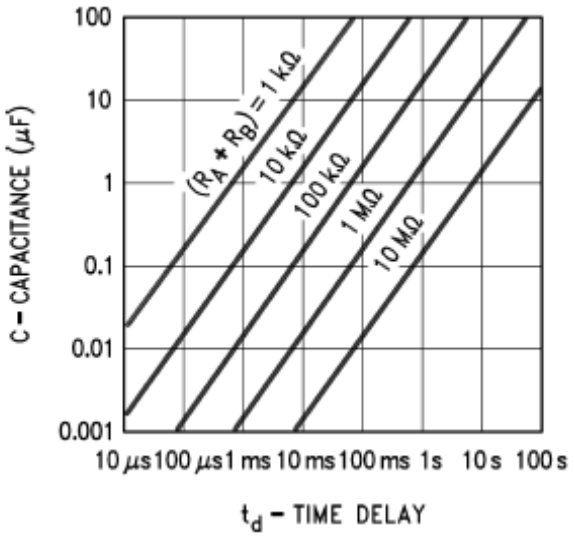
Εικόνα 32: Συνδεσμολογία του 555 για λειτουργία one-shot [14]

Η λειτουργία του 555 στην παραπάνω διάταξη περιγράφεται από το διάγραμμα:



Εικόνα 33: Διάγραμμα χρονισμού του 555 σε λειτουργία one-shot [14]

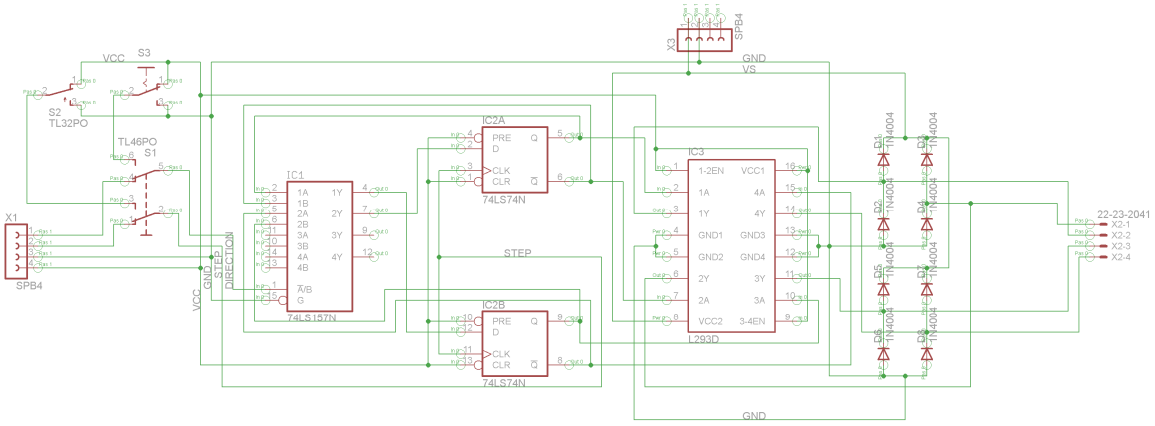
Το πρώτο σήμα είναι το σήμα εισόδου που τροφοδοτεί τον ακροδέκτη 2, το δεύτερο είναι το σήμα εξόδου που παράγεται στον ακροδέκτη 3 και το τρίτο η τάση στα άκρα του πυκνωτή C που φαίνεται στην προηγούμενη εικόνα. Η διάρκεια του παλμού εξόδου καθορίζεται από την τιμή της αντίστασης R_A και της χωρητικότητας του πυκνωτή C, με τη σχέση που περιγράφεται στο διάγραμμα:



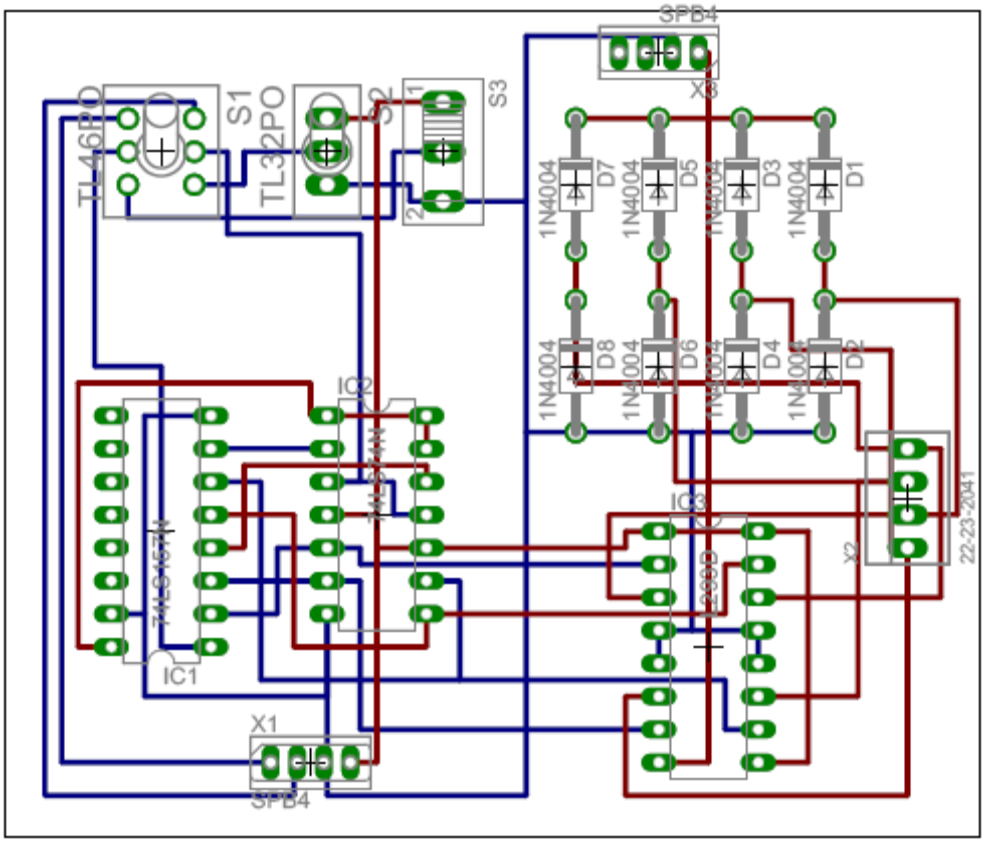
Εικόνα 34: Σχέση αντίστασης, χωρητικότητας και χρόνου καθυστέρησης [14]

Διάταξη

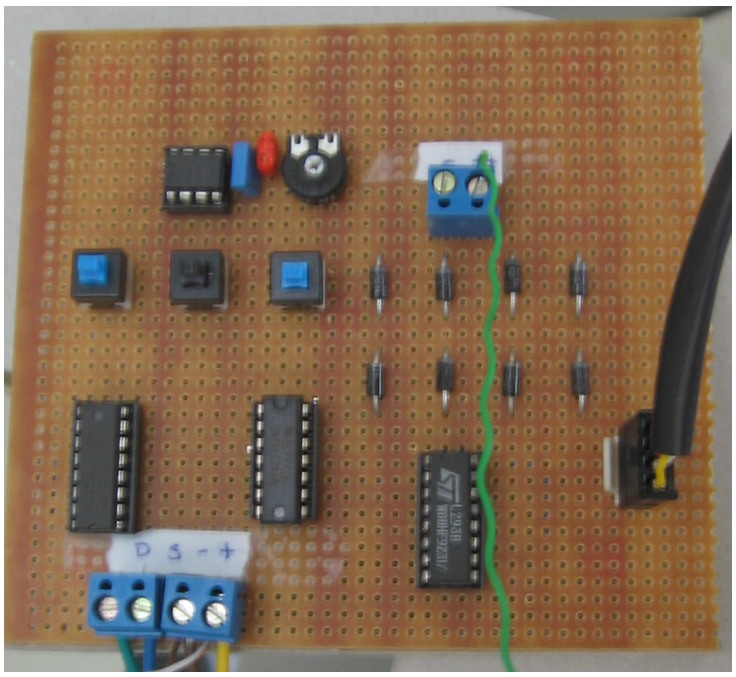
Τα σχέδια του driver δίνονται στη συνέχεια.



Εικόνα 35: Λειτουργικό διάγραμμα του driver του κινητήρα



Εικόνα 36: Σχέδιο κατασκευής της διάτρητης πλακέτας του driver του κινητήρα



Εικόνα 37: Ο driver του κινητήρα

ΛΟΓΙΣΜΙΚΟ

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ

Προγραμματισμός του μικροεπεξεργαστή

Για τη λειτουργία των κόμβων απαιτείται ο προγραμματισμός του μικροεπεξεργαστή. Η διαδικασία αυτή περιλαμβάνει την αποθήκευση του προγράμματος που έχει αναπτυχθεί και είναι διαθέσιμο σε δυαδική ή δεκαεξαδική μορφή στην ενσωματωμένη μνήμη Flash του μικροεπεξεργαστή. Η μεταφορά μπορεί να γίνει βασικά με δύο τρόπους:

- Το σύστημα ISP (In-System Programming)

  Ο LPC2129 διαθέτει ενσωματωμένο λογισμικό ελέγχου εκκίνησης (bootloader) το οποίο ενεργοποιείται όταν ο ακροδέκτης P0.14 βρίσκεται σε χαμηλό δυναμικό κατά την επανεκκίνηση. Το λογισμικό αυτό επιτρέπει την πρόσβαση στη μνήμη flash και τον προγραμματισμό της μέσω κατάλληλων εντολών διαμέσου της σειριακής θύρας UART0.
- Τη θύρα JTAG

  Ο προγραμματισμός της ενσωματωμένης μνήμης flash μπορεί να γίνει και με χρήση της θύρας JTAG ακολουθώντας κάποια συγκεκριμένα βήματα, αλλά στην εφαρμογή θα χρησιμοποιηθεί η πρώτη μέθοδος.

Η χρήση του συστήματος ISP γίνεται με ιδιαίτερη ευκολία στην ολοκληρωμένη πλακέτα (header board) λόγω της σύνδεσης του συνδετήρα DSUB-9 που χρησιμοποιείται για το πρωτόκολλο RS-232 με τους κατάλληλους ακροδέκτες του LPC2129. Συγκεκριμένα, οι ακροδέκτες 7 και 4 του DSUB-9 (σήματα RTS και DTR) βρίσκονται συνδεδεμένοι, διαμέσου των jumper J6 και J7 αντίστοιχα, ο πρώτος με τον ακροδέκτη P0.14 του μικροεπεξεργαστή και ο δεύτερος με την είσοδο του ρυθμιστή τάσης που προκαλεί την επανεκκίνηση του συστήματος. Με τον τρόπο αυτό, είναι δυνατός ο έλεγχος τόσο της επανεκκίνησης όσο και της κατάστασης του P0.14 του LPC2129 και επομένως η ενεργοποίηση του λογισμικού ελέγχου εκκίνησης αποκλειστικά μέσω της σειριακής θύρας.

Ο έλεγχος της θύρας RS-232 γίνεται από τον προσωπικό υπολογιστή. Οι λεπτομέρειες της επικοινωνίας που επιτρέπουν τη μεταφορά του δεκαεξαδικού αρχείου που αποτελεί το πρόγραμμα ρυθμίζονται αυτόματα από το λογισμικό που εκτελείται στον υπολογιστή. Η Philips προσφέρει δωρεάν το εργαλείο LPC2000 Flash Utility, το οποίο

διαθέτει παραθυρικό περιβάλλον και επιτρέπει τη μεταφορά ενός δεκαεξαδικού αρχείου που βρίσκεται στο σκληρό δίσκο του υπολογιστή. Παράλληλα, ο χρήστης μπορεί να επιλέξει τις λεπτομέρειες του πρωτοκόλλου RS-232 ώστε να είναι συμβατό με τις αντίστοιχες ρυθμίσεις που έχουν γίνει στον LPC2129 και εκτός του προγραμματισμού, ο χρήστης μπορεί να διαβάσει κατασκευαστικές λεπτομέρειες όπως τον τύπο του μικροεπεξεργαστή (το εργαλείο είναι συμβατό με όλη τη σειρά LPC2000) και την ταυτότητα του συγκεκριμένου chip.

Ένα δεύτερο εργαλείο που είναι διαθέσιμο δωρεάν είναι το LPC21ISP, το οποίο και χρησιμοποιείται στην εφαρμογή. Ο κύριος λόγος για αυτό είναι ότι εκτός του προγραμματισμού της μνήμης, το εργαλείο προσφέρει και λειτουργία τερματικού για την αποστολή και λήψη δεδομένων από τη σειριακή θύρα. Ως περιβάλλον προγραμματισμού του μικροελεγκτή παρέχει όλες τις λειτουργίες που περιγράφηκαν και νωρίτερα αλλά χωρίς το γραφικό περιβάλλον. Ως τερματικό, είναι ικανό να τυπώνει την οθόνη κάθε δεδομένο που δέχεται η θύρα RS-232 του υπολογιστή καθώς και να αποστέλλει σε αυτήν κάθε δεδομένο που πληκτρολογεί ο χρήστης. Η λειτουργία του τερματικού χρησιμοποιείται στην εφαρμογή για τον έλεγχο και την επικοινωνία μεταξύ του χρήστη και του κόμβου 1.

Οι επιλογές εκτέλεσης του LPC21ISP φαίνονται στην παρακάτω εικόνα:

```
C:\>lpc21isp

Portable command line ISP for Philips LPC2000 family and
Version 1.31                Analog Devices ADUC 70xx
Compiled for Windows: Nov 27 2005 15:10:40
Copyright (c) by Martin Maurer, 2003-2005  Email: Martin.Maurer@clibb.de
Portions Copyright (c) by Aeolus Development 2004
                http://www.aeolusdevelopment.com

Syntax:  lpc21isp [Options] file comport baudrate Oscillator_in_kHz

Example: lpc21isp test.hex com1 115200 14746

Options: -bin           for uploading binary file
         -hex           for uploading file in intel hex format (default)
         -term          for starting terminal after upload
         -termonly      for starting terminal without an upload
         -detectonly    detect only used LPC chiptype (PHILIPSARM only)
         -debug         for creating a lot of debug infos
         -control       for controlling RS232 lines for easier booting
                        (Reset = DTR, EnableBootLoader = RTS)
         -logfile       for enabling logging of terminal output to lpc21isp.log
         -ADARM         for downloading to an Analog Devices
                        ARM microcontroller ADUC70xx
         -PHILIPSARM    for downloading to a microcontroller from
                        Philips LPC2000 family (default)
```

Εικόνα 38: Επιλογές εκτέλεσης του LPC21ISP [13]

Ο χρήστης έχει τη δυνατότητα ελέγχου των ρυθμίσεων της θύρας, επιλογής για το αν θα ενεργοποιηθεί ο έλεγχος της επανεκκίνησης και του λογισμικού ελέγχου εκκίνησης (bootloader), γεγονός απαραίτητο για τη μεταφορά του προγράμματος αλλά όχι επιθυμητό όταν απαιτείται η χρήση των αντίστοιχων σημάτων της θύρας κατά την κανονική εκτέλεση του προγράμματος, κάποιες επιλογές για καταγραφή πληροφοριών χρήσιμων για την αποσφαλμάτωση της διαδικασίας και επιλογή για το αν θα γίνει μεταφορά προγράμματος ή μόνο λειτουργία τερματικού.

ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Γενικά

Για την ανάπτυξη του προγράμματος που θα αποθηκευτεί και θα εκτελείται στο μικροεπεξεργαστή απαιτείται κάποιο εργαλείο συγγραφής του κώδικα σε γλώσσα προγραμματισμού υψηλού επιπέδου που θα μεταφραστεί στη συνέχεια στη γλώσσα μηχανής του LPC2129, δηλαδή σε εντολές ARM, σε ένα δεκαεξαδικό αρχείο ικανό να αποθηκευτεί και να εκτελεστεί από τον πυρήνα. Εξαιτίας της ευρείας διάδοσης των επεξεργαστών ARM υπάρχουν πλήθος μεταφραστών για διάφορες γλώσσες προγραμματισμού, τόσο αυτόνομα όσο και ως μέρος συνολικότερων εργαλείων ανάπτυξης εφαρμογών. Για τη συγκεκριμένη εφαρμογή επιλέχθηκε ο μεταφραστής GCC (GNU Compiler Collection) για πυρήνες ARM, στην έκδοση 4.1.1, ο οποίος έχει χαμηλότερες επιδόσεις ως προς την πυκνότητα και την ταχύτητα του κώδικα που παράγει σε σύγκριση με άλλους αλλά διατίθεται δωρεάν, γεγονός για το οποίο επιλέχθηκε έναντι των μεταφραστών που περιλαμβάνονται σε ολοκληρωμένα εργαλεία το κόστος των οποίων είναι απαγορευτικό. Για τη συγγραφή του κώδικα επιλέχθηκε ο επεξεργαστής κειμένου Programmers Notepad στην έκδοση 2.0.6.1. Ο συγκεκριμένος, πέρα από τις βασικές λειτουργίας επεξεργασίας κειμένου, διαθέτει δυνατότητες διαχείρισης projects, δηλαδή σύνολα αρχείων που αποτελούν τον πηγαίο κώδικα της εφαρμογής, καθώς και δυνατότητα ενσωμάτωσης του εργαλείου make του GCC ώστε η διαδικασία παραγωγής του δεκαεξαδικού αρχείου που θα εκτελεστεί από τον μικροεπεξεργαστή να γίνεται αυτόματα. Η ανάπτυξη του κώδικα για τη συγκεκριμένη εφαρμογή γίνεται σε C, γλώσσα που προσφέρει προγραμματισμό τόσο σε υψηλό επίπεδο όσο και σε χαμηλότερο, μέχρι ακόμα και τη συμπερίληψη αυτούσιων εντολών μηχανής στην assembly του ARM.

GNU Compiler Collection

Η μετάφραση του προγράμματος που έχει γραφτεί σε C απαιτεί διάφορα στάδια, την υλοποίηση των οποίων αναλαμβάνουν επιμέρους κομμάτια του λογισμικού που απαρτίζει τη συλλογή GCC. Εν συντομία, τα μέρη αυτά είναι:

- Ο assembler, που παράγει δυαδικό κώδικα από κώδικα σε assembly και τον τοποθετεί σε ένα αρχείο-στόχο (object file)
- Ο preprocessor, που επεξεργάζεται τα απαιτούμενα αρχεία επικεφαλίδων (header files)
- Ο linker, που συνδέει τον κώδικα με συγκεκριμένες διευθύνσεις, συνδυάζοντας το αρχείο εκκίνησης και τις βιβλιοθήκες που απαιτούνται για τη δημιουργία μιας δυαδικής εκτελέσιμης εικόνας

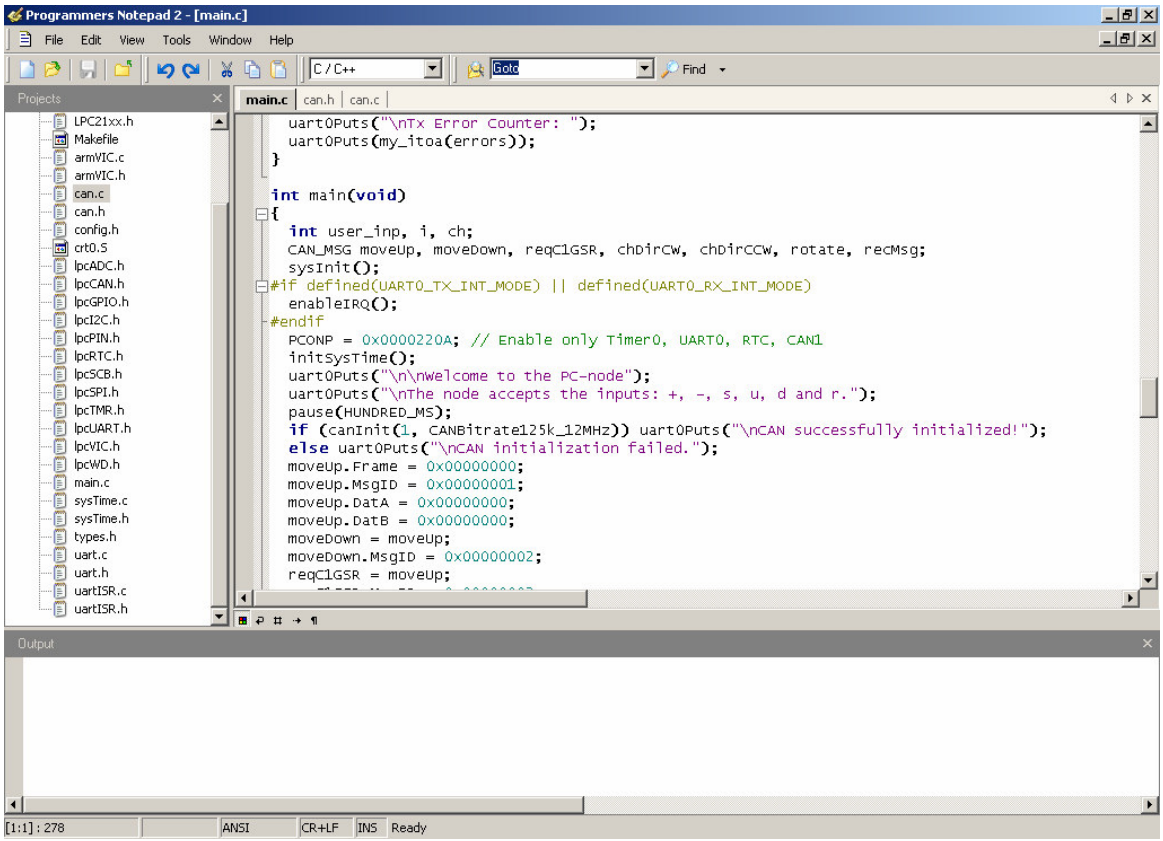
Η χρήση όλων των επιμέρους εργαλείων είναι διάφανη προς το χρήστη με την εκτέλεση του εργαλείου make. Η λειτουργία του καθορίζεται από τα στοιχεία που περιλαμβάνονται στο αρχείο makefile, όπου περιγράφονται οι σχέσεις ανάμεσα στα αρχεία που αποτελούν τον πηγαίο κώδικα του προγράμματος. Το εργαλείο φροντίζει να περάσει τα αρχεία αυτά στη συλλογή του GCC έτσι ώστε η τελική έξοδος να είναι ένα εκτελέσιμο δεκαεξαδικό αρχείο, έτοιμο για αποθήκευση και εκτέλεση στο μικροεπεξεργαστή.

Programmers Notepad

Ο επεξεργαστής κειμένου που χρησιμοποιήθηκε διατίθεται επίσης δωρεάν, ενώ ο πηγαίος του κώδικας είναι ανοικτός. Περιλαμβάνει όλες τις χρήσιμες λειτουργίες ενός επεξεργαστή κειμένου για ανάπτυξη κώδικα, όπως λειτουργίες αντιγραφής και επικόλλησης, εύρεσης κι αντικατάστασης, μετατροπής χαρακτήρων, χειρισμού γραμμών καθώς και αυτόματη μορφοποίηση ανάλογα με τη γλώσσα προγραμματισμού που χρησιμοποιείται, όπως αναγνώριση δεσμευμένων λέξεων και δομών τύπου if-else και βρόχων, για την καλύτερη οπτικοποίηση του κώδικα. Επιπλέον, έχει την ικανότητα να διαχειρίζεται σύνολα αρχείων ως projects, τα οποία συνήθως περιλαμβάνουν τα αρχεία πηγαίου κώδικα και τα αρχεία ρυθμίσεων του μεταφραστή, ο οποίος καλείται με το απλό πάτημα ενός κλικ πάλι μέσα από το Programmers Notepad. Τέλος, στο διαδίκτυο υπάρχουν πολλές έτοιμες εφαρμογές για τους επεξεργαστές της σειράς LPC2000 που περιλαμβάνουν εξαιρετικά χρήσιμα αρχεία επικεφαλίδων και αρχεία ρυθμίσεων, οι

οποίες είναι σε μορφή project χειρίσιμου από το συγκεκριμένο επεξεργαστή κειμένου και μπορούν εύκολα να χρησιμοποιηθούν ως πρότυπα.

Μία εικόνα από το περιβάλλον του Programmers Notepad δίνεται στη συνέχεια.



Εικόνα 39: Περιβάλλον του Programmers Notepad

ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ

Γενικά

Ο πηγαίος κώδικας των προγραμμάτων των δύο κόμβων CAN αποτελείται από ένα σύνολο αρχείων. Από αυτά, το κυρίως πρόγραμμα βρίσκεται στο αρχείο main.c, ενώ τα υπόλοιπα που συναποτελούν το project στα πλαίσια του Programmers Notepad είναι αρχεία επικεφαλίδων, στα οποία ορίζονται κάποια χαρακτηριστικά σχετικά με το αντίστοιχο περιφερειακό του μικροεπεξεργαστή, αρχεία κώδικα, όπου ορίζονται οι συναρτήσεις που εκτελούν τις βασικές λειτουργίες για κάθε περιφερειακό καθώς και τα αρχεία οδηγιών προς τον assembler, το linker και το εργαλείο make του GCC. Από τα παραπάνω αρχεία, όλα εκτός από το main.c και το can.c είναι κοινά και πάρθηκαν αυτούσια από τα παραδείγματα που συνοδεύουν το Programmers Notepad. Συγκεκριμένα, αυτά που χρησιμοποιήθηκαν έχουν σχεδιαστεί και διατίθενται στο διαδίκτυο ελεύθερα για κάθε χρήση από την ομάδα R O Software και επιλέχθηκαν εξαιτίας της εξαιρετικά δομημένης μορφής που έχουν. Μόνη εξαίρεση είναι το αρχείο επικεφαλίδων can.h, το οποίο αναπτύχθηκε εξ ολοκλήρου στα πλαίσια της εργασίας αυτής ακολουθώντας τη μορφή των υπολοίπων αντίστοιχων αρχείων.

Το σύνολο του κώδικα των κοινών αρχείων που χρησιμοποιήθηκαν δίνεται στο Παράρτημα Α. Στο σημείο αυτό θα γίνει μια σύντομη ανάλυση σε ένα χαρακτηριστικό παράδειγμα, με δεδομένο ότι με παρόμοιο τρόπο αναλύονται και τα υπόλοιπα.

ΚΟΙΝΑ ΑΡΧΕΙΑ

Ακολουθώντας τη συμπερίληψη των αρχείων όπως δηλώνεται στο main.c, αρχικά βρίσκεται το αρχείο **types.h**, όπου ορίζονται κάποιοι χρήσιμοι τύποι μεταβλητών διαφορετικών μεγεθών συσχετίζοντας συμβολικά ονόματα αντίστοιχα του επιθυμητού αριθμού bits με τους τύπους που υπάρχουν διαθέσιμοι στη standard C. Στη συνέχεια συμπεριλαμβάνεται το αρχείο **LPC21xx.h**, το οποίο με τη σειρά του φροντίζει για τη συμπερίληψη των αρχείων επικεφαλίδων που σχετίζονται με τα ενσωματωμένα στο μικροεπεξεργαστή περιφερειακά. Επιπλέον, στο αρχείο αυτό ορίζονται συμβολικά ονόματα για καθέναν από τους καταχωρητές διαμέσου των οποίων υλοποιείται η λειτουργία των περιφερειακών. Το αρχείο LPC21xx.h τροποποιήθηκε στα πλαίσια της

διπλωματικής εργασίας έτσι ώστε να συμπεριλαμβάνονται οι ορισμοί των καταχωρητών των ελεγκτών CAN. Παρόλο που ο LPC2129 διαθέτει μόνο 2 τέτοιους ελεγκτές, στο αρχείο προβλέφθηκαν 4, χαρακτηριστικό που ισχύει για άλλους μικροεπεξεργαστές της σειράς LPC21xx, έτσι ώστε η χρήση του αρχείου να παραμένει γενική για ολόκληρη τη σειρά σε περίπτωση που κάποιος επόμενος χρήστης θελήσει να υλοποιήσει εφαρμογή με κάποιον επεξεργαστή με περισσότερους των 2 ελεγκτές CAN.

Για να γίνει σαφής ο τρόπος με τον οποίο οι ορισμοί που βρίσκονται στο αρχείο LPC21xx.h σε συνδυασμό με τους ορισμούς των αρχείων lpcxxx.h (πχ. lpcCAN.h) επιτρέπουν την πρόσβαση στους καταχωρητές του LPC2129, θα αναλυθεί ο κώδικας που αφορά τους ελεγκτές CAN. Απαιτείται μια ανασκόπηση τόσο στους καταχωρητές του συστήματος CAN όσο και στο σύστημα μνήμης. Έχει αναφερθεί ότι ένα τμήμα του χάρτη μνήμης είναι αφιερωμένο στους καταχωρητές των περιφερειακών. Το τμήμα αυτό είναι εσωτερικά χωρισμένο ανά περιφερειακό και κάθε κομμάτι που προκύπτει επίσης αναλύεται ανά καταχωρητή, με αποτέλεσμα όλοι οι καταχωρητές ενός υποσυστήματος, όπως το υπό μελέτη CAN, να αντιστοιχίζονται μοναδικά με συνεχόμενες διευθύνσεις στη μνήμη. Όπως έχει επίσης αναφερθεί, οι διευθύνσεις αυτές είναι ευθυγραμμισμένες σε λέξεις των 32-bit, ανεξάρτητα από το πραγματικό μέγεθος του καταχωρητή.

Αναλύοντας λοιπόν βήμα-βήμα το **lpcCAN.h** (βλ. Παράρτημα Α), στην αρχή ελέγχεται αν το αρχείο έχει ήδη συμπεριληφθεί ώστε να αποτραπεί ο νέος ορισμός των καταχωρητών. Με τη μέθοδο αυτή υπερισχύει η πρώτη εμφάνιση του αρχείου, αν υπάρξουν και επόμενες. Στη συνέχεια ορίζονται οι δομές που αντιστοιχούν στα διάφορα υποσυστήματα που συναποτελούν το σύστημα CAN, με πρώτο τη μνήμη του Φίλτρου Αποδοχής. Από τον τρόπο με τον οποίο γίνεται η διαχείριση της μνήμης στον compiler της standard C, είναι γνωστό ότι οι μεταβλητές ή οι σταθερές που έχουν τύπο δομής (struct) αποθηκεύονται σε διαδοχικές διευθύνσεις. Η δομή, επομένως, που ονομάζεται canAfRAM_t έχει ως στοιχεία έναν καταχωρητή μήκους 32 bit με όνομα start, έναν καταχωρητή μήκους 32 bit με όνομα end και στο ενδιάμεσο ένα "κενό", δηλαδή 510 συνεχόμενες θέσεις μνήμης μήκους 32 bit. Με τον τρόπο αυτό, η δομή που μόλις ορίστηκε καταλαμβάνει 32 x 512 = 16384 bits = 2048 Bytes = 2 kB μνήμης, ακριβώς όση προβλέπεται από τα τεχνικά χαρακτηριστικά του LPC2129. Η επόμενη δομή αντιστοιχεί στους καταχωρητές λειτουργίας του Φίλτρου Αποδοχής. Ονομάζεται canAfRegs_t και περιέχει 8 καταχωρητές των 32 bit, οι οποίοι ορίζονται με βάση τη σειρά και τις αντίστοιχες διευθύνσεις μνήμης που δίνονται στα τεχνικά χαρακτηριστικά του LPC2129.

Η δομή canCntrlRegs_t περιλαμβάνει τους κεντρικούς κοινούς καταχωρητές του συστήματος CAN, μήκους 32 bit. Τέλος, η δομή canRegs_t περιλαμβάνει όλους τους καταχωρητές που ελέγχουν κάθε έναν από τους ελεγκτές CAN αλλά ορίζεται μία μόνο φορά, αφού το σύνολο αυτό είναι ίδιο για κάθε ελεγκτή και απλά βρίσκεται σε διαφορετικό σημείο στη μνήμη για κάθε ελεγκτή. Όπως και προηγουμένως, ο ορισμός των καταχωρητών γίνεται με συγκεκριμένη σειρά ώστε μετά τη δήλωση των αντίστοιχων μεταβλητών ή σταθερών, το κάθε συμβολικό όνομα καταχωρητή να αντιστοιχίζεται πράγματι στη θέση μνήμης που είναι συσχετισμένη εκ κατασκευής με αυτόν.

Για τη σωστή τοποθέτηση των παραπάνω δομών στο χάρτη μνήμης καθώς και την αναφορά στους καταχωρητές με εύκολα συμβολικά ονόματα, ίδια με αυτά που ορίζονται στα τεχνικά χαρακτηριστικά του μικροεπεξεργαστή φροντίζει το αρχείο LPC21xx.h. Αναλυτικότερα, στο τμήμα του κώδικα με τίτλο «CAN Controllers» ορίζονται για κάθε υποσύστημα του CAN πρώτα συμβολικά ονόματα (με κεφαλαία γράμματα) δεικτών σε δομές αντίστοιχες του υποσυστήματος αυτού, οι οποίοι δείχνουν σε διευθύνσεις μνήμης που καθορίζονται συγκεκριμένα. Σε αυτό ακριβώς το σημείο είναι που παρέχεται πρόσβαση στις επιθυμητές θέσεις μνήμης σύμφωνα με τα χαρακτηριστικά του χάρτη μνήμης για το σύστημα CAN. Εφόσον ο δείκτης δείχνει σε δομή που είναι τοποθετημένη στη σωστή θέση μνήμης, οι ορισμοί των συμβολικών ονομάτων των καταχωρητών που ακολουθούν είναι υπεύθυνοι για τη συσχέτιση σε επίπεδο προγραμματιστή του ονόματος κάθε καταχωρητή (με κεφαλαία γράμματα) με τη διεύθυνση μνήμης που πραγματικά του αντιστοιχεί. Ύστερα από τους ορισμούς των κοινών στοιχείων βρίσκονται οι ορισμοί των καταχωρητών κάθε ελεγκτή, όπου φαίνεται και γιατί δεν υπάρχει ανάγκη ξεχωριστής δομής για τον καθένα.

Η διαδικασία που περιγράφηκε παραπάνω είναι κοινή για όλα τα περιφερειακά του μικροελεγκτή. Η μόνη διαφοροποίηση που παρατηρείται σε ορισμένες περιπτώσεις, όπως στο **lpcUART.h**, είναι ότι στις δομές μπορεί να περιλαμβάνονται καταχωρητές μήκους 8 bit οπότε και απαιτείται να ακολουθούνται από 3 "κενές" θέσεις μήκους 8 bit ώστε να ικανοποιείται ο περιορισμός της ευθυγράμμισης του συστήματος μνήμης σε λέξεις των 32 bit. Μια ακόμα λειτουργία που δεν υπάρχει στο lpcCAN.h αλλά παρατηρείται στο lpcUART.h είναι ότι στο τελευταίο βρίσκονται και ορισμοί σταθερών που μπορούν να διευκολύνουν το χειρισμό καταχωρητών.

Συνεχίζοντας την ανάλυση των αρχείων που συμπεριλαμβάνονται στο main.c, ακολουθεί το αρχείο **config.h**, όπου βρίσκονται διάφοροι ορισμοί σταθερών που

καθορίζουν τα χαρακτηριστικά λειτουργίας του επεξεργαστή και των περιφερειακών του. Ορίζεται λοιπόν η σταθερά HOST_BAUD που θα χρησιμοποιηθεί στην αρχικοποίηση του συστήματος UART ως ρυθμός μεταφοράς δεδομένων. Εδώ έχει επιλεχθεί συχνότητα 38,4kbps. Στη συνέχεια ορίζονται οι σταθερές που σχετίζονται με το σύστημα χρονισμού. Συγκεκριμένα δηλώνεται η συχνότητα του κρυστάλλου, η τιμή του πολλαπλασιαστή PLL, η τιμή του ρολογιού του πυρήνα που προκύπτει ως το γινόμενο των παραπάνω, ο διαιρέτης VPB και η συνεπαγόμενη συχνότητα του ρολογιού των περιφεριακών του διαύλου VPB. Το τμήμα αυτό καταλήγει με μια σειρά από ελέγχους για την καταλληλότητα των παραπάνω τιμών, σύμφωνα με όσα ορίζονται στα τεχνικά χαρακτηριστικά του LPC2129. Τέλος, δίνονται κάποιοι ορισμοί για το σύστημα των εξόδων γενικής χρήσης, οι οποίοι όμως θα μείνουν αχρησιμοποίητοι στη συγκεκριμένη εφαρμογή.

Το επόμενο αρχείο είναι το **armVIC.h**, το οποίο προσφέρει τις δηλώσεις που απαιτούνται για τη ρύθμιση και τον έλεγχο του συστήματος διακοπών του πυρήνα. Παρόλο που το σύστημα αυτό δεν χρησιμοποιείται άμεσα στην εφαρμογή, οπότε και δεν θα αναλυθεί εδώ, η συμπερίληψη του αρχείου επιβάλλεται λόγω της χρήσης μερικών από τα στοιχεία του από το σύστημα UART. Όπως και στα επόμενα παρόμοια σε λειτουργία και δομή αρχεία που θα αναλυθούν, το αρχείο επικεφαλίδων armVIC.h συνοδεύεται από το αρχείο πηγαίου κώδικα armVIC.c όπου βρίσκονται οι εντολές που αποτελούν το πραγματικό σώμα των συναρτήσεων που δηλώνονται στο αρχείο επικεφαλίδων.

UART

Στη συνέχεια των συμπεριλήψεων βρίσκεται το αρχείο **UART.h**, που συνδυάζεται με το **UART.c** για να δημιουργήσει το πλαίσιο λειτουργίας του συστήματος UART. Στο αρχείο επικεφαλίδων ορίζονται σταθερές που χρησιμοποιούνται για να επιλεγούν τα χαρακτηριστικά λειτουργίας του συστήματος και οι συναρτήσεις που υλοποιούν τη διεπαφή του UART με το προγραμματιστικό περιβάλλον.

Αναλυτικότερα, η πρώτη συνάρτηση (uart0init) είναι η αρχικοποίηση του συστήματος. Δέχεται ως ορίσματα τον επιθυμητό ρυθμό μετάδοσης δεδομένων (BAUD rate), τον τρόπο λειτουργίας και τις ρυθμίσεις του FIFO υποσυστήματος. Η μορφή των ορισμάτων ακολουθεί συγκεκριμένους κανόνες που απαιτούν γνώση του κώδικα της συνάρτησης. Ως τμήμα της διεπαφής όμως, ο προγραμματιστής που καλεί τη συνάρτηση δεν θα έπρεπε να γνωρίζει τις λεπτομέρειες της υλοποίησης, επομένως οι ορισμοί που

προηγούνται φροντίζουν ώστε οι διάφορες συνήθεις επιλογές να αντιστοιχίζονται σε εύχρηστα μνημονικά ονόματα. Το πρώτο όρισμα προκύπτει μέσα από τον ορισμό της μακροεντολής UART_BAUD() ως συνάρτηση του επιθυμητού ρυθμού μετάδοσης δεδομένων σε μονάδες που χρησιμοποιούνται κατά κόρον (bits per second). Το δεύτερο πρέπει να είναι συμβατό με τον τρόπο λειτουργίας του καταχωρητή U0LCR οπότε για την ευχρηστία της συνάρτησης έχουν ήδη οριστεί τα δεδομένα που αντιστοιχούν σε συγκεκριμένα σύνολα ρυθμίσεων σε ορισμούς που έχουν ευνόητα μνημονικά ονόματα και με τη σειρά τους κάνουν χρήση των ορισμών που έχουν γίνει στο αρχείο lpcUART.h. Παρόμοια ισχύουν και για το τρίτο όρισμα, με αποτέλεσμα η κλήση να καταλήγει να απαιτεί μια απλή γνώση των τρόπων λειτουργίας του συστήματος UART και των ονομάτων των αντίστοιχων σταθερών. Η εκτέλεση της συνάρτησης ρυθμίζει τους καταχωρητές του UART με τρόπο ώστε, αφού ενεργοποιηθεί, να λειτουργεί όπως έχει επιλεχθεί σύμφωνα με τα ορίσματα. Στο σημείο αυτό γίνεται και η χρήση του συστήματος διακοπών, ορίζοντας τη συνάρτηση uart0ISR, ο κώδικας της οποίας βρίσκεται στο αρχείο **uart0ISR.c**, ως συνάρτηση εξυπηρέτησης των διακοπών που προκαλούνται από τον ελεγκτή UART. Ταυτόχρονα, αλλάζει τις τιμές κάποιων εξωτερικών μεταβλητών που επιτρέπουν στις υπόλοιπες συναρτήσεις του αρχείου να γνωρίζουν την κατάσταση του συστήματος.

Η κλήση της επόμενης συνάρτησης (uart0Putch) προκαλεί την τοποθέτηση του χαρακτήρα που δίνεται ως όρισμα στην ουρά αποστολής δεδομένων και επιστρέφει τον ίδιο το χαρακτήρα σε περίπτωση επιτυχίας ενώ τιμή -1 σε αντίθετη περίπτωση. Η βοηθητική συνάρτηση uart0Space επιστρέφει το διαθέσιμο χώρο στην ουρά αποστολής και χρησιμοποιείται μαζί με την προηγούμενη στην uart0Write η οποία εγγράφει στην ουρά το καθοριζόμενο πλήθος χαρακτήρων από ένα δοσμένο σύνολο. Αντίστοιχη λειτουργία επιτελεί η uart0Puts η οποία εγγράφει μια ολόκληρη συμβολοσειρά και είναι αυτή που χρησιμοποιείται στον κύριο κώδικα της εφαρμογής. Η συνάρτηση uart0TxEmpty επιστρέφει την κατάσταση των καταχωρητών δεδομένων αποστολής και η uart0TxFlush καθαρίζει την ουρά αποστολής αποτρέποντας την αποστολή των δεδομένων που δεν έχουν ήδη σταλεί. Για τη λήψη δεδομενων υπάρχει μία μόνο συνάρτηση, η uart0Getch, η οποία ελέγχει την ουρά λήψης δεδομένων και επιστρέφει το χαρακτήρα που έχει ληφθεί ή τιμή -1 αν δεν υπάρχει διαθέσιμος χαρακτήρας. Τέλος, στο αρχείο περιλαμβάνονται οι ίδιες συναρτήσεις και για το δεύτερο ελεγκτή (UART1), του οποίου δεν γίνεται χρήση στη συγκεκριμένη εφαρμογή.

Η λειτουργία των παραπάνω συναρτήσεων βασίζεται στην ύπαρξη δύο πινάκων χαρακτήρων ορισμένου στο αρχείο επικεφαλίδων μήκους, με ονόματα uart0_rx_buffer και uart0_tx_buffer, που υλοποιούν τις ουρές λήψης και αποστολής δεδομένων αντίστοιχα. Κάθε ένας από τους πίνακες αυτούς συνοδεύεται από δύο ακεραίους αριθμούς, έναν που ορίζει την αρχή (extract) και έναν το τέλος (insert) της ουράς. Οι χαρακτήρες που λαμβάνονται τοποθετούνται στο τέλος και επεξεργάζονται από το κύριο πρόγραμμα από την αρχή της ουράς, ενώ οι χαρακτήρες που αποστέλλονται τοποθετούνται στο τέλος και αποστέλλονται από την αρχή της ουράς.

Η διαδικασία που εκτελείται κατά τη λήψη και την αποστολή χαρακτήρων μέσω της σειριακής θύρας UART στη συγκεκριμένη εφαρμογή περιγράφεται στον κώδικα της συνάρτησης εξυπηρέτησης διακοπών uart0ISR. Όταν ληφθεί ή αποσταλεί ένας χαρακτήρας ενεργοποιείται η αντίστοιχη διακοπή, ο τύπος της οποίας ελέγχεται μέσω του καταχωρητή U0IIR. Σε περίπτωση λήψης, ο χαρακτήρας είναι τοποθετημένος στον καταχωρητή U0RBR. Η συνάρτηση τον αντιγράφει στην επόμενη διαθέσιμη θέση στην ουρά λήψης και ενημερώνει κατάλληλα τους δείκτες της ουράς. Ο προγραμματιστής οφείλει στη συνέχεια να καλέσει τη συνάρτηση uart0Getch ώστε να επεξεργαστεί το χαρακτήρα που ελήφθει και να τον αφαιρέσει από την ουρά απελευθερώνοντας χώρο για τους επόμενους που θα ληφθούν. Αντίστοιχα, για την αποστολή ενός χαρακτήρα, η κλήση της uart0Putch ελέγχει αν τη στιγμή εκείνη ο ελεγκτής βρίσκεται σε διαδικασία αποστολής, οπότε και τοποθετεί τον προς αποστολή χαρακτήρα στο τέλος της ουράς αποστολής ή είναι ανενεργός, οπότε τοποθετεί απ' ευθείας το χαρακτήρα στον καταχωρητή U0THR ώστε να αποσταλεί με την πρώτη ευκαιρία, δηλαδή μόλις ελευθερωθεί η γραμμή.

Χρονιστής

Το τελευταίο κοινό αρχείο που υπάρχει στον πηγαίο κώδικα και των δύο κόμβων CAN είναι το **sysTime.h**. Στο αρχείο επικεφαλίδων αυτό ορίζονται χρονικές σταθερές ως συναρτήσεις της συχνότητας του ρολογιού και οι συναρτήσεις που διαμορφώνουν τη διεπαφή του συστήματος με τον προγραμματιστή. Ο κώδικας των συναρτήσεων αυτών βρίσκεται στο αντίστοιχο πηγαίο αρχείο **sysTime.c**. Για τον ορισμό των σταθερών απαιτούνται κάποιοι υπολογισμοί σχετικοί με τη συχνότητα του ρολογιού του πυρήνα καθώς και μια υπενθύμιση του συστήματος χρονιστών του LPC2129.

Η συχνότητα των περιφερειακών pclk προκύπτει ως το πηλίκο της συχνότητας του πυρήνα cclk δια το διαιρέτη VPB. Από τις ρυθμίσεις που ορίζονται στο αρχείο config.h, ο διαιρέτης αυτός με το μνημονικό όνομα PBSD ορίζεται στην τιμή 2, επομένως η συχνότητα του pclk είναι 30MHz. Οι παλμοί του ρολογιού αυτού αυξάνουν κατά 1 το μετρητή κλίμακας PC ενώ η τιμή του καταχωρητή επιλογής κλίμακας PR καθορίζει τη σχέση του PC με τον κεντρικό μετρητή TC. Για να προκύψει ένα βολικό ελάχιστο χρονικό διάστημα μετά το πέρας του οποίου αυξάνεται η τιμή του κεντρικού μετρητή, επιλέγεται η τιμή 3 για τον PR. Με τον τρόπο αυτό, ο κεντρικός μετρητής TC αυξάνεται με συχνότητα 10MHz, δηλαδή με περίοδο 100nsec.

Η συνάρτηση initSysTime αναλαμβάνει την αρχικοποίηση του χρονιστή timer0 ο οποίος είναι ο μόνος που χρησιμοποιείται. Η υλοποίηση δεν κάνει χρήση των δυνατοτήτων διακοπών του συστήματος ούτε της επικοινωνίας με εξωτερικούς ακροδέκτες καθώς δεν υπάρχει αντίστοιχη ανάγκη. Αφού ρυθμιστούν οι παράμετροι λειτουργίας του συστήματος, γίνεται η εκκίνηση του μετρητή και αρχικοποιείται στην τιμή 0 η εξωτερική μεταβλητή sysTics. Η συνάρτηση getSysTICs επιστρέφει την τρέχουσα ώρα συστήματος μετρημένη σε TICs, μονάδα χρόνου ίση με 100nsec, όπως αναφέρθηκε προηγουμένως. Η συνάρτηση getElapsedSysTICs επιστρέφει τη διαφορά μεταξύ της τρέχουσας ώρας και μιας αρχικής στιγμής που δίνεται ως όρισμα, πάντα σε μονάδες TICs. Τέλος, η συνάρτηση pause, που είναι και ο βασικός λόγος συμπερίληψης του συστήματος χρονισμού στην εφαρμογή, σταματάει τη λειτουργία της ΚΜΕ κάνοντας χρήση της συνάρτησης WDOG του υποσυστήματος watchdog ώστε να συνεχίσει φυσιολογικά η εκτέλεση εντολών ύστερα από την παύση. Η διάρκεια της παύσης δίνεται σε μονάδες TICs ως όρισμα κατά την κλήση.

Αξίζει να σημειωθεί μια ιδιαιτερότητα της υλοποίησης που σχετίζεται με τη λειτουργία του χρονιστή. Ο κεντρικός μετρητής TC έχει μήκος 32 bit, επομένως μπορεί να λάβει τιμές έως 2^{32}-1 = 4294967295. Σε μονάδες TICs αυτό αντιστοιχεί σε περίπου 430 sec. Αυτό σημαίνει ότι κάθε 430 δευτερόλεπτα, ο κεντρικός μετρητής TC μηδενίζεται και ξεκινάει τη μέτρηση από την αρχή. Για να αντιμετωπιστεί το ζήτημα αυτό, η συνάρτηση getSysTICs χρησιμοποιεί την εξωτερική μεταβλητή sysTICs ως συσσωρευτή (accumulator), προσθέτοντας κάθε φορά τα TICs που έχουν διανυθεί από την τελευταία φορά που έγινε κλήση της συνάρτησης. Προϋπόθεση για τη σωστή λειτουργία της υλοποίησης είναι η συνάρτηση να καλείται τουλάχιστον μία φορά κάθε περίπου 430 sec. Η προϋπόθεση αυτή τηρείται στην εφαρμογή.

Αρχεία οδηγιών GCC

Στα projects που αναπτύσσονται για κάθε έναν από τους δύο κόμβους περιλαμβάνονται ορισμένα αρχεία εκτός των αρχείων που περιέχουν τον πηγαίο κώδικα. Αυτά καθορίζουν τις λεπτομέρειες εκτέλεσης των εργαλείων του GCC. Το make λειτουργεί με βάση το αρχείο **makefile**, στο οποίο ορίζονται ο τύπος του πυρήνα και το μοντέλο του μικροεπεξεργαστή, αν το πρόγραμμα θα αποθηκευτεί στη μνήμη RAM ή ROM, ο τύπος του αρχείου που θα προκύψει από τη μετάφραση του κώδικα, το σύνολο των αρχείων που περιέχουν τον πηγαίο κώδικα και πρέπει να μεταφραστούν, το αρχείο οδηγιών του assembler, το επίπεδο βελτιστοποίησης που εδώ επιλέχθηκε με σκοπό την ελαχιστοποίηση του μεγέθους του κώδικα που προκύπτει και πολλές ακόμα επιλογές σχετικά με τους απαιτούμενους υποφακέλους, τα μηνύματα επικοινωνίας προς το χρήστη, τις επιλογές του assembler κ.ά.

Το αρχείο **crt0.s** περιέχει κώδικα σε assembly που φροντίζει για τον ορισμό του διανύσματος διακοπών, την αντιγραφή του κώδικα στη ROM αν απαιτείται, την υλοποίηση μιας συνάρτησης που χρησιμοποιείται για την κατασκευή των αντικειμένων αν ο κώδικας είναι γραμμένος στην αντικειμενοστραφή C++ και τέλος μεταφέρει την εκτέλεση στον κύριο κώδικα καλώντας τη συνάρτηση main.

Το αρχείο **LPC2129-ROM.ld** χρησιμοποιείται από τον linker του GCC όταν έχει επιλεχθεί εκτέλεση από τη ROM ενώ το αντίστοιχο με όνομα -RAM χρησιμοποιείται για την εκτέλεση από τη RAM. Με βάση τους ορισμούς που περιλαμβάνονται στο αρχείο, ο linker τοποθετεί τα διάφορα κομμάτια κώδικα που έχουν προκύψει από τα προηγούμενα στάδια της μετάφρασης στις κατάλληλες θέσεις ώστε το τελικό δεκαεξαδικό αρχείο που θα αποθηκευτεί στη μνήμη του μικροεπεξεργαστή να εκτελείται με τον επιθυμητό τρόπο.

CAN

Η διεπαφή με τους ελεγκτές CAN του μικροεπεξεργαστή υλοποιείται μέσω ενός συνόλου συναρτήσεων, με τρόπο αντίστοιχο με τα προηγούμενα περιφερειακά που περιγράφηκαν (UART, χρονιστές). Οι συναρτήσεις αυτές ορίζονται στο αρχείο επικεφαλίδων can.h και ο κώδικάς τους βρίσκεται στο πηγαίο αρχείο can.c. Το αρχείο επικεφαλίδων, εκτός του ορισμού των συναρτήσεων αναλαμβάνει και τον ορισμό των δύο σταθερών που αντιστοιχούν στους ρυθμούς μετάδοσης 125 και 250 kbps και χρησιμοποιούνται ως ορίσματα στη συνάρτηση αρχικοποίησης του συστήματος. Επιπλέον, ορίζει τη δομή

CAN_MSG που αναπαριστά ένα μήνυμα CAN, είτε βασικής είτε εκτεταμένης δομής, που αποτελείται από τέσσερα πεδία μήκους 32 bit: το πεδίο Frame, το πεδίο MsgID και τα πεδία DatA και DatB.

Το πεδίο Frame αντιστοιχεί στους καταχωρητές CANRFS και CANTFI, δηλαδή τους καταχωρητές πλαισίου των ληφθέντων και των προς αποστολή μηνυμάτων. Κοιτώντας τη δομή του CANRFS, τα 10 πρώτα bits καθορίζουν τη θέση που έχει το αναγνωριστικό του μηνύματος που λήφθηκε στον πίνακα του Φίλτρου Αποδοχής, δεδομένου ότι αυτό είναι ενεργοποιημένο, γεγονός που ισχύει στην εφαρμογή. Τα αντίστοιχα bits των προς αποστολή μηνυμάτων δεν παίζουν ρόλο, δεδομένου ότι χρησιμοποιούνται μόνο όταν γίνεται χρήση όλων των buffers αποστολής, ενώ στη συγκεκριμένη εφαρμογή χρησιμοποιείται αποκλειστικά ο πρώτος. Τα bits 16 έως 19 καθορίζουν το πλήθος των bytes δεδομένων που περιέχονται στο μήνυμα. Το bit 30 είναι μονάδα όταν το μήνυμα είναι μήνυμα αίτησης αποστολής και μηδέν όταν είναι κανονικό μήνυμα δεδομένων. Το bit 31 είναι μονάδα όταν η δομή του μηνύματος είναι εκτεταμένη ενώ μηδέν όταν είναι βασική. Το πεδίο MsgID περιέχει το αναγνωριστικό του μηνύματος και είναι 11 bits όταν το μήνυμα είναι βασικής δομής και 29 bits όταν είναι εκτεταμένης. Αντιστοιχεί στους καταχωρητές CANRID και CANTID για τη λήψη και την αποστολή αντίστοιχα. Τα πεδία DatA και DatB περιέχουν αυτά καθεαυτά τα δεδομένα του μηνύματος, ανάλογα με το πλήθος των bytes που έχει καθοριστεί στο DLC (Data Length Counter - bits 19:16) και αντιστοιχούν στους καταχωρητές CANRDA, CANRDB, CANTDA και CANTDB.

Το αρχείο can.c υλοποιεί τις τρεις συναρτήσεις που αποτελούν τη διεπαφή με τον ελεγκτή CAN. Προτού αναλυθούν αυτές, απαιτείται μια επισκόπηση των μηνυμάτων που χρησιμοποιούνται στην εφαρμογή. Πρόκειται για 6 μηνύματα και ένα βοηθητικό για την εκκίνηση του συστήματος. Το βοηθητικό έχει αναγνωριστικό 0 και δεν περιέχει δεδομένα, απλά αποστέλλεται πριν την αποστολή του πρώτου μηνύματος ώστε να αντιμετωπιστεί το πρόβλημα που περιγράφεται στο errata sheet κατά τη μετάβαση από την κατάσταση επανεκκίνησης στην κατάσταση κανονικής λειτουργίας των ελεγκτών CAN. Τα υπόλοιπα 6 μηνύματα έχουν αναγνωριστικά από 1 έως 6 και το καθένα αποστέλλεται μόνο από τον έναν από τους δύο κόμβους, ικανοποιώντας τον περιορισμό που προκύπτει από τον τρόπο διαχείρισης της διαιτησίας του διαύλου και έχει περιγραφεί παραπάνω. Όπως είναι προφανές, το πλήθος των μηνυμάτων μπορεί να ικανοποιηθεί από αναγνωριστικά μήκους 11 bits και επομένως η βασική δομή είναι ικανοποιητική.

Το μήνυμα 1, δηλαδή αυτό που έχει αναγνωριστικό 1, συνοδεύεται από 1 byte δεδομένων και αποστέλλεται από τον Κεντρικό Κόμβο προκειμένου ο ανελκυστήρας να μεταβεί στον όροφο ο αριθμός του οποίοι βρίσκεται στα σχετικά δεδομένα. Το μήνυμα 2 δεν συνοδεύεται από δεδομένα και προκαλεί το σταμάτημα του ανελκυστήρα. Το μήνυμα 3 επίσης δεν περιέχει δεδομένα και αποστέλλεται από τον Κεντρικό Κόμβο ως αίτηση για την αποστολή των περιεχομένων του καταχωρητή C1GSR του Κόμβου Ανελκυστήρα. Το μήνυμα 4 αποστέλλεται είναι ίδιο με μορφή με το προηγούμενο και αποτελεί αίτηση της κατάστασης του ανελκυστήρα. Το μήνυμα 5 αποστέλλεται από τον Κόμβο Ανελκυστήρα και στα 4 bytes δεδομένων που το συνοδεύουν δίνονται τα περιεχόμενα του καταχωρητή C1GSR του κόμβου αυτού. Το μήνυμα 6 περιέχει την κατάσταση του ανελκυστήρα, κωδικοποιημένη στα 8 bits που το συνοδεύουν ως εξής: τα bits 0 έως 2 περιέχουν τον αριθμό του ορόφου στον οποίο βρίσκεται ο ανελκυστήρας και είναι τόσα καθώς απαιτείται η γνώση της θέσης του ανελυστήρα που μπορεί να είναι και μεταξύ ορόφων, επομένως οι αριθμοί 0, 2 και 4 σημαίνουν ότι ο ανελκυστήρας βρίσκεται στο ισόγειο, τον πρώτο και το δεύτερο όροφο αντίστοιχα και οι αριθμοί 1 και 3 μεταξύ ισογείο ακι πρώτου και μεταξύ πρώτου και δευτέρου ορόφου. Το bit 3 είναι μονάδα εάν ο ανελκυστήρας είναι γεμάτος, το bit 4 εάν είναι σταματημένος, το bit 5 εάν ο διακόπτης κινδύνου είναι ενεργοποιημένος και τα bits 6 και 7 δηλώνουν τον όροφο στον οποίο κατευθύνεται ο ανελκυστήρας.

Η πρώτη συνάρτηση της διεπαφής είναι η canInit που φροντίζει για την αρχικοποίηση του περιφερειακού επιστρέφοντας μονάδα σε περίπτωση επιτυχίας. Το περιφερειακό ρυθμίζεται έτσι ώστε να μην χρησιμοποιούνται διακοπές και, ακολουθώντας τις προτάσεις της κατασκευάστριας Philips στο errata sheet, δεν κάνει χρήση της λειτουργίας FullCAN, των καταστάσεων μειωμένης λειτουργικότητας και κατανάλωσης ενέργειας και των πολλαπλών buffers αποστολής. Για την αποστολή ενεργοποιείται μόνο ο buffer 1 και όλα τα μηνύματα στέλνονται με την εντολή Self Reception Request αντί της Transmission Request, με αποτέλεσμα ο κόμβος να λαμβάνει και ο ίδιος τα μηνύματα που στέλνει. Τα μηνύματα που στέλνει ο ίδιος, μολαταύτα, δεν γίνονται δεκτά από το Φίλτρο Αποδοχής οπότε, λειτουργικά, παράγεται το ίδιο αποτέλεσμα.

Τα ορίσματα που δίνονται στην canInit κατά την κλήση της είναι ο αριθμός του ελεγκτή CAN που θα ενεργοποιηθεί και η ταχύτητα μετάδοσης μεταφοράς δεδομένων. Στη συγκεκριμένη εφαρμογή υποστηρίζονται μόνο ο πρώτος ελεγκτής και οι ρυθμοί

μετάδοσης 125 και 250kbps, ρυθμίσεις που ελέγχονται πριν την εκτέλεση του κυρίως σώματος των εντολών. Στη συνέχεια, ορίζοντας τις τιμές των καταχωρητών ελέγχου του συστήματος CAN, αυτό τοποθετείται σε κατάσταση επανεκκίνησης, με κενό καταχωρητή κατάστασης, ρυθμό μετάδοσης τον επιλεγμένο και τις διακοπές αποενεργοποιημένες. Στην κατάσταση αυτή είναι εφικτή η ρύθμιση των παραμέτρων του Φίλτρου Αποδοχής.

Το τελευταίο αρχικά απενεργοποιείται και με τη βοήθεια ενός δείκτη που τοποθετείται στην αρχή του πρώτου πίνακα του συστήματος του φίλτρου, ο οποίος περιέχει τα μεμονωμένα αναγνωριστικά που γίνονται αποδεκτά. Εφόσον τα μηνύματα στην εφαρμογή κάνουν χρήση της βασικής δομής, το αναγνωριστικό τους είναι μήκους 11 bits και η τοποθέτηση των δεκτών μεμονωμένων αναγνωριστικών γίνεται έχοντας 2 αναγνωριστικά σε κάθε σειρά του πίνακα. Ορίζονται λοιπόν ως δεκτά για τον ελεγκτή με αριθμό 1 τα αναγνωριστικά που αντιστοιχούν στα μηνύματα που στέλνονται από τον άλλο κόμβο από αυτόν στον οποίο αποθηκεύεται το εκάστοτε πρόγραμμα. Για το λόγο αυτό, η συνάρτηση διαφοροποιείται σε αυτό το σημείο στους δύο κόμβους: στον Κεντρικό Κόμβο τα αναγνωριστικά που ορίζονται ως δεκτά είναι τα 5 και 6, που αντιστοιχούν στα μηνύματα που αποστέλλονται από τον Κόμβο Ανελκυστήρα, ενώ στον Κόμβο Ανελκυστήρα τα δεκτά αναγνωριστικά είναι τα 1, 2, 3 και 4. Σε κάθε περίπτωση, στο τέλος του πρώτου πίνακα που μόλις ρυθμίστηκε τοποθετούνται δύο αναγνωριστικά που είναι όμως απενεργοποιημένα, ακολουθώντας τις συμβουλές του κατασκευαστή του LPC2129. Οι υπόλοιποι πίνακες του Φίλτρου Αποδοχής τίθενται κενοί ορίζοντας ως διεύθυνση αρχής και τέλους την ίδια. Η τοποθέτηση δύο απενεργοποιημένων αναγνωριστικών αντίστοιχων με τα προηγούμενα απαιτείται και στο τέλος της περιοχής μνήμης που είναι αφιερωμένη στο Φίλτρο Αποδοχής. Μετά τις παραπάνω ρυθμίσεις, το Φίλτρο Αποδοχής ενεργοποιείται και ο ελεγκτής τοποθετείται σε κατάσταση κανονικής λειτουργίας. Τέλος, προκειμένου να αντιμετωπιστεί η δυσλειτουργία που εμφανίζεται κατά τη μετάβαση από κατάσταση επανεκκίνησης στην κανονική, ετοιμάζεται και αποστέλλεται ένα μήνυμα με αναγνωριστικό 0, χωρίς δεδομένα και με συγκεκριμένες εντολές αποστολής.

Η συνάρτηση που είναι υπεύθυνη για την αποστολή ενός μηνύματος CAN ονομάζεται canSend. Δέχεται ως ορίσματα τον αριθμό του ελεγκτή μέσω του οποίου θα γίνει η αποστολή και ένα δείκτη στη δομή που περιέχει το προς αποστολή μήνυμα ενώ επιστρέφει μονάδα σε περίπτωση επιτυχίας. Η υλοποίηση της εφαρμογής κάνει χρήση μόνο του πρώτου ελεγκτή, γεγονός που ελέγχεται αρχικά. Στη συνέχεια μεταφέρει τα

πεδία της δομής του μηνύματος στους αντίστοιχους καταχωρητές αποστολής και δίνει εντολή αποστολής διαμέσου του πρώτου ελεγκτή διατηρώντας τον περιορισμό χρήσης της εντολής SRR (Self-Reception Request).

Με δεδομένο ότι δεν γίνεται χρήση διακοπών, η ανάγνωση ενός μηνύματος που έχει ληφθεί απαιτεί την κλήση της συνάρτησης canReceive από το κυρίως πρόγραμμα σε τακτά χρονικά διαστήματα. Η συνάρτηση αυτή δέχεται ως ορίσματα τον αριθμό του ελεγκτή που θα ελεγχθεί για ληφθέν μήνυμα CAN, ο οποίος επιτρέπεται να είναι μόνο ο 1, και έναν δείκτη στη δομή όπου θα αποθηκευτεί το μήνυμα προς περαιτέρω επεξεργασία. Όταν κληθεί, η συνάρτηση ελέγχει για ύπαρξη ληφθέντος μηνύματος μέσω ανάγνωσης του bit RBS του καταχωρητή C1GSR. Αν αυτό είναι μηδενικό, τότε ο ελεγκτής δεν έχει λάβει κάποιο νέο μήνυμα και η συνάρτηση επιστρέφει την τιμή μηδέν. Διαφορετικά, το νέο μήνυμα αντιγράφεται στη δομή που έχει περαστεί ως όρισμα από τους καταχωρητές λήψης, δίνεται εντολή αποδέσμευσης του buffer λήψης ώστε να είναι δυνατή η λήψη επόμενου μηνύματος και η συνάρτηση επιστρέφει την τιμή 1.

ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ

Κεντρικός Κόμβος

Ο Κεντρικός Κόμβος είναι υπεύθυνος για την επικοινωνία, μέσω της σειριακής θύρας, με το χρήστη και την ανταλλαγή των κατάλληλων μηνυμάτων, μέσω του διαύλου CAN, με τον Κόμβο Ανελκυστήρα. Ο πηγαίος του κώδικας περιλαμβάνει εκτός της κύριας συνάρτησης main και μερικές βοηθητικές. Η αρχικοποίηση του συστήματος υλοποιείται με τις συναρτήσεις lowInit και sysInit, με την πρώτη να καλείται στην αρχή της δεύτερης και τη δεύτερη να καλείται στην αρχή της main. Με σειρά λοιπόν εκτέλεσης, η lowInit αρχικά περνάει τις τιμές M του πολλαπλασιαστή και P του διαιρέτη του κυκλώματος PLL στον καταχωρητή PLLCFG χρησιμοποιώντας τις σταθερές που έχουν οριστεί στο αρχείο config.h και στη συνέχεια ενεργοποιεί το κύκλωμα δίνοντας διαδοχικά στον καταχωρητή PLLFEED τις τιμές που απαιτούνται από το μικροεπεξεργαστή. Το υπόλοιπο μέρος της συνάρτησης ρυθμίζει και ενεργοποιεί το MAM και το διαιρέτη VPB, πάντα με χρήση σταθερών των οποίων οι τιμές έχουν οριστεί στο config.h. Η συνάρτηση sysInit, αφού καλέσει τη lowInit, ενεργοποιεί το σύστημα χαρτογράφησης μνήμης μεταφέροντας τα

διανύσματα διακοπών στο σημείο που απαιτείται ανάλογα με τον αν έχει επιλεχθεί εκτέλεση του κώδικα από τη μνήμη RAM ή τη flash. Αρχικοποιεί το σύστημα διακοπών και καλεί τη συνάρτηση αρχικοποιήσης του UART, που αναλύθηκε προηγουμένως.

Στα πλαίσια της εκτύπωσης χαρακτήρων στη σειριακή θύρα, απαιτείται κάποια συνάρτηση μετατροπής αριθμών σε συμβολοσειρές. Από τις απαιτήσεις της υλοποίησης, είναι γνωστό ότι οι αριθμοί αυτοί που πρόκειται να εκτυπωθούν είναι το μέγιστο τριψήφιοι, επομένως και η συνάρτηση μετατροπής λειτουργεί αντίστοιχα. Δέχεται ως όρισμα τον αριθμό σε τύπο ακεραίου και επιστρέφει μια συμβολοσειρά, δηλαδή ένα δείκτη σε πίνακα χαρακτήρων, το τελευταίο στοιχείο του οποίου οφείλει να είναι {0} δηλώνοντας το τέλος της συμβολοσειράς. Κατά την κλήση της, η συνάρτησης ελέγχει αν η είσοδος είναι ο αριθμός 0, οπότε και αποθηκεύει το χαρακτήρα '0' στην προτελευταία θέση του πίνακα και επιστρέφει δείκτη στα 2 τελευταία στοιχεία, δηλαδή το χαρακτήρα '0' και το χαρακτήρα τέλους συμβολοσειράς. Ο χαρακτήρας '0' λαμβάνεται ως το πρώτο στοιχείο της συμβολοσειράς "0", λύση που διαμορφώνει τον κώδικα ανεξάρτητο του συστήματος στο οποίο μεταφράζεται και εκτελείται. Αν η είσοδος είναι οποιοσδήποτε άλλος αριθμός, γνωρίζοντας πάντα ότι οι αριθμοί είναι θετικοί ακέραιοι, εκτελείται ένας βρόχος που επιλέγει το χαρακτήρα που αντιστοιχεί σε κάθε ψηφίο μέσα από μια συμβολοσειρά που περιλαμβάνει όλους τους χαρακτήρες αριθμών με τη σειρά. Στο τέλος του βρόχου επιστρέφεται δείκτης σε συμβολοσειρά που περιέχει μόνο τα στοιχεία του πίνακα χαρακτήρων που χρειάζονται.

Η ανάλυση των βοηθητικών συναρτήσεων τελειώνει με τη συνάρτηση printGSR, η οποία καλείται όταν απαιτείται η εκτύπωση στη σειριακή των περιεχομένων του καταχωρητή GSR, είτε του Κεντρικού είτε του Κόμβου Ανελκυστήρα. Ελέγχονται ένα-ένα τα bit του καταχωρητή και για κάθε ένα εκτυπώνεται το όνομά του και η τιμή του, ανάλογα με το αν είναι μονάδα ή μηδέν, ώστε το αποτέλεσμα να είναι ευανάγνωστο από το χρήστη.

Η συνάρτηση main αρχικοποιεί το σύστημα καλώντως τη sysInit και ενεργοποιεί τις διακοπές ώστε να χρησιμοποιηθούν για τη λειτουργία του UART. Ρυθμίζει τον καταχωρητή PCONP ενεργοποιώντας μόνο τα αναγκαία περιφερειακά, δηλαδή το χρονιστή 0, τον ελεγκτή 0 του UART, το ρολόι πραγματικού χρόνου (Real Time Clock) και τον ελεγκτή 1 του CAN. Καλεί τη συνάρτηση αρχικοποίησης του συστήματος χρονισμού και τυπώνει στη σειριακή ένα μήνυμα εκκίνησης που αναφέρει τους χαρακτήρες που γίνονται δεκτοί ως είσοδος από το χρήστη. Μετά από μια μικρή παύση,

καλείται η συνάρτηση αρχικοποίησης του ελεγκτή CAN και τυπώνεται το αντίστοιχο μήνυμα επιτυχίας ή αποτυχίας. Στη συνέχεια ετοιμάζονται τα μηνύματα που αποστέλλονται από τον κόμβο και αποθηκεύονται στις αντίστοιχες μεταβλητές τύπου μηνύματος CAN.

Η κύρια λειτουργία του μικροεπεξεργαστή καθορίζεται στον αέναο κόμβο που αποτελεί τον κορμό της κύριας συνάρτησης. Σε κάθε εκτέλεσή του ελέγχεται ο ελεγκτής UART για την ύπαρξη νέου χαρακτήρα στη σειριακή θύρα. Αν υπάρχει νέος χαρακτήρας, ελέγχεται ποιος είναι αυτός ώστε να εκτελεστούν οι αντίστοιχες εντολές. Αν ο χρήστης έστειλε το χαρακτήρα 'm', στη σειριακή τυπώνεται το μήνυμα "Move To Floor" και ο χρήστης καλείται να πιέσει τον αριθμό του ορόφου στον οποίο επιθυμεί να μετακινηθεί ο ανελκυστήρα. Αν η είσοδος από το χρήστη είναι αποδεκτή, αποστέλλεται το αντίστοιχο μήνυμα στο δίαυλο CAN. Η επιτυχία ή αποτυχία της αποστολής προκαλεί την εκτύπωση του αντίστοιχου μηνύματος. Ο χαρακτήρας 'h' προκαλεί την αποστολή του μηνύματος "Stop". Η λήψη του χαρακτήρα 'u' προκαλεί την κλήση της συνάρτησης printGSR και την εκτύπωση των περιεχομένων του καταχωρητή GSR του Κεντρικού Κόμβου. Η είσοδος από το χρήστη των χαρακτήρων 's' και 'e' προκαλούν την αποστολή των μηνυμάτων αίτησης της κατάστασης του ελεγκτή CAN του Κόμβου Ανελκυστήρα και της κατάστασης του ανελκυστήρα αντίστοιχα. Σε κάθε άλλο χαρακτήρα, ο Κεντρικός Κόμβος απλά τυπώνει το μήνυμα "No Command".

Εκτός του ελέγχου της σειριακής θύρας RS-232 για είσοδο δεδομένων από το χρήστη, ο Κόμβος ελέγχει και το δίαυλο CAN για ύπαρξη μηνυμάτων. Όπως έχει καθοριστεί στη συνάρτηση αρχικοποίησης στο αρχείο can.c, το Φίλτρο Αποδοχής του Κεντρικού Κόμβου αφήνει να περάσουν μόνο τα μηνύματα με αναγνωριστικά 5 και 6, δηλαδή το μήνυμα με τον καταχωρητή GSR του Κόμβου Ανελκυστήρα και το μήνυμα με την κατάσταση του ανελκυστήρα. Τα αναγνωριστικά των μηνυμάτων αυτών έχουν τοποθετηθεί στις δύο πρώτες θέσεις του Πίνακα Αποδοχής, επομένως όταν λαμβάνεται κάποιο μήνυμα με ένα από αυτά τα αναγνωριστικά, τα πρώτα 10 bits του πεδίου frame περιέχουν τον αριθμό της θέσης στην οποία βρέθηκε το αναγνωριστικό του μηνύματος που λήφθηκε.

Σε κάθε εκτέλεση του αέναου βρόχου, λοιπόν, γίνεται έλεγχος του ελεγκτή CAN για ύπαρξη νέου μηνύματος. Αν υπάρχει, ελέγχεται το πεδίο frame αυτού και διακρίνονται δύο περιπτώσεις:

- Αν ο αριθμός του είναι 0, τότε καλείται η συνάρτηση printGSR περνώντας ως όρισμα τα 4 bytes δεδομένων που λήφθηκαν και έχουν τα περιεχόμενα του 32-bit καταχωρητή GSR του Κόμβου Ανελκυστήρα.
- Αν ο αριθμός του είναι 1, τότε τυπώνονται μια σειρά από μηνύματα που περιγράφουν την τρέχουσα κατάσταση του ανελκυστήρα, σύμφωνα με τη δομή του byte που συνοδεύει το μήνυμα αυτό. Επιπλέον, ενημερώνονται οι μεταβλητές isEmpty και onStop που είναι μονάδα όταν ο ανελκυστήρας είναι κενός και σταματημένος αντίστοιχα ή μηδέν σε αντίθετη περίπτωση.

Καθώς ο Κεντρικός Κόμβος ελέγχει και την κατάσταση των πλήκτρων κλήσης από τον κάθε όροφο, στη συνέχεια του αέναου βρόχου αποστέλλονται τα σχετικά μηνύματα, αφού έχει ελεγχθεί ότι ο ανελκυστήρας είναι σταματημένος. Στη συγκεκριμένη υλοποίηση, εξυπηρετείται μόνο η τελευταία κλήση που έγινε καθώς δεν υπάρχει κάποιο σύστημα μνήμης και άρα τα πατήματα που γίνονται όσο ο ανελκυστήρας κινείται αγνοούνται.

Στο τελευταίο τμήμα του βρόχου υλοποιείται μία συνήθης λειτουργία των ανελκυστήρων: η επιστροφή του θαλάμου στο ισόγειο ύστερα από μια περίοδο αδράνειας. Ο Κεντρικός Κόμβος ελέγχει αν όντως ο ανελκυστήρας έχει παραμείνει κενός και σταματημένος για περισσότερο από δύο λεπτά και σε αυτήν την περίπτωση αποστέλλει το μήνυμα CAN που προκαλεί τη μετάβασή του στο ισόγειο. Προκειμένου να μην σταματούν οι υπόλοιπες λειτουργίες του κόμβου, γίνεται χρήση των μεταβλητών timerStarted και startTime. Αυτές υλοποιούν ένα σύστημα επανεκκίνησης του χρονομετρητή όποτε ο ανελκυστήρας ενεργοποιηθεί.

Κόμβος Ανελκυστήρα

Ο Κόμβος Ανελκυστήρα είναι υπεύθυνος για τον έλεγχο του 7-segment display, των πλήκτρων ελέγχου που βρίσκονται στο θάλαμο, του μαγνητικού αισθητήρα προσέγγισης ορόφου και τη ρύθμιση του επιπέδου τάσης του ακροδέκτη P0.2 που αποτελεί το σήμα ορισμού της φοράς περιστροφής του βηματικού κινητήρα καθώς και την παραγωγή των παλμών στον ακροδέκτη P0.3 που προκαλούν την περιστροφή κατά βήματα του κινητήρα. Ταυτόχρονα, ελέγχει το δίαυλο CAN για τυχόν μηνύματα από τον Κεντρικό Κόμβο και τον ενημερώνει για κάθε αλλαγή της κατάστασης του ανελκυστήρα.

Για την τέλεση των λειτουργιών του, ο Κόμβος Ανελκυστήρα κάνει χρήση τριών βοηθητικών συναρτήσεων και τριών κύριων καθώς και ορισμένων γενικών μεταβλητών

που περιγράφουν την κατάσταση του θαλάμου. Η πρώτη από τις βοηθητικές συναρτήσεις, με τη σειρά που ορίζονται, ονομάζεται updateDisplay και χωρίς να δέχεται ορίσματα ή να επιστρέφει τιμή ανανεώνει την ένδειξη του 7-segmet display σύμγωνα με την τιμή της μεταβλητής elevFloor, που περιέχει την τρέχουσα θέση του θαλάμου. Η μεταβλητή αυτή παίρνει τις τιμές 0 έως και 4 ώστε να περιλαμβάνει και τις θέσεις μεταξύ ορόφων. Η ρύθμιση της ένδειξης γίνεται ορίζοντας σε μονάδες τις ψηφιακές εξόδους του μικροεπεξεργαστή που αντιστοιχούν στα τμήματα που σχηματίζουν τους εκάστοτε αριθμούς.

Η επόμενη βοηθητική συνάρτηση ονομάζεται statusEvaluation και επιστρέφει έναν ακέραιο στη μορφή που απαιτείται από το μήνυμα που περιέχει την τρέχουσα κατάσταση του ανελκυστήρα. Η τρίτη βοηθητική συνάρτηση ονομάζεται keypadEvaluation και επιστρέφει την τιμή του πλήκτρου ορόφου που πατήθηκε ή την τιμή -1 εάν δεν έχει πατηθεί κάποιο πλήκτρο.

Οι τρεις επόμενες συναρτήσεις που ορίζονται καλούνται στον αέναο βρόχο που εκτελείται. Η πρώτη ονομάζεται checkCAN και αναλαμβάνει να ελέγξει εάν έχει ληφθεί κάποιο μήνυμα από το δίαυλο CAN και να εκτελέσει τις αντίστοιχες ενέργειες. Συγκεκριμένα, αν το μήνυμα που λήφθηκε είναι εντολή μετάβασης σε όροφο, τότε ουσιαστικά αναιρείται η όποια εντολή είναι τρέχουσα και ανάλογα με τη θέση του ανελκυστήρα, αυτός τίθεται σε κίνηση προς τον όροφο που έχει ζητηθεί από τον Κεντρικό Κόμβο. Με τον τρόπο αυτό, ο Κεντρικός Κόμβος τοποθετείται πρώτος σε ιεραρχία, αφού οι εντολές του εκτελούνται άμεσα. Αν λήφθηκε το μήνυμα παύσης κίνησης, ο ανελκυστήρα ακινητοποιείται άμεσα. Αν έχει ληφθεί κάποιο από τα μηνύματα αίτησης κατάστασης, είτε του GSR του ελεγκτή CAN του κόμβου είτε του ανελκυστήρα, τα αντίστοιχα δεδομένα των μηνυμάτων ενημερώνονται και πραγματοποιείται η αποστολή. Σε περίπτωση μη ύπαρξης μηνύματος στο δίαυλο, η συνάρτηση επιστρέφει.

Οι δύο άλλες συναρτήσεις περιγράφουν τη λειτουργία του ανελκυστήρα όταν αυτός βρίσκεται σε στάση ή σε κίνηση, κατάσταση που καθορίζεται από την τιμή της μεταβλητής onStop. Αν λοιπόν ο θάλαμος είναι σταματημένος, ελέγχεται η κατάσταση των ψηφιακών εισόδων LOAD και ALARM και εφόσον ο ανελκυστήρας δεν είναι κενός ούτε σε κατάσταση κινδύνου, ελέγχονται τα πλήκτρα ορόφου. Αν πατήθηκε κάποιο πλήκτρο, τότε γίνεται σύγκριση της τρέχουσας θέσης του θαλάμου με τη ζητούμενη θέση και ξεκινάει η κίνηση του ανελκυστήρα προς την επιθυμητή θέση. Αν δεν έχει πατηθεί κάποιο πλήκτρο, η συνάρτηση απλά επιστρέφει. Αν ο ανελκυστήρας βρίσκεται σε

κίνηση, εκτελείται η άλλη συνάρτηση, στην οποία αρχικά τίθεται η φορά περιστροφής του κινητήρα ανάλογα με το εάν η επιθυμητή θέση είναι πάνω ή κάτω από την τρέχουσα. Ο πρώτος μικρός βρόχος εκτελείται μετακινώντας τον ανελκυστήρα προς την επιθυμητή κατεύθυνση έως ότου ο μαγνητικός αισθητήρας χάσει την επαφή του με τον τρέχοντα όροφο, λειτουργία που απαιτείται όταν η εκκίνηση γίνεται από κάποιον όροφο και όχι από ενδιάμεση θέση. Εφόσον ο θάλαμος έχει φύγει από τον προηγούμενο όροφο, η εκτέλεση προκαλεί την περιστροφή του κινητήρα κατά ένα μόνο βήμα και στη συνέχεια ελέγχει εάν ο θάλαμος έχει προσεγγίσει τον επόμενο όροφο. Αν προσεγγίστηκε κάποιος όροφος, ελέγχεται εάν αυτός είναι ο επιθυμητός, οπότε και σταματάει η κίνηση διαφορετικά ανανεώνεται η ένδειξη και η κίνηση συνεχίζεται κανονικά. Αν δεν έχει προσεγγιστεί κάποιος όροφος η συνάρτηση επιστρέφει ώστε να υλοποιηθούν και οι υπόλοιπες λειτουργίες του κόμβου, οι οποίες εκτελούνται παράλληλα.

Η αρχικοποίηση του συστήματος γίνεται με τον ίδιο τρόπο που περιγράφηκε παραπάνω για τον Κεντρικό Κόμβο. Η διαφοροποίηση έγκειται στο γεγονός ότι το σύστημα UART δεν χρησιμοποιείται καθόλου, επομένως τόσο τα αρχεία επικεφαλίδων και κώδικα που σχετίζονται με αυτό όσο και αυτό καθεαυτό το περιφερειακό απενεργοποιείται. Ενεργοποιημένα παραμένουν, διαμέσου της τιμής του καταχωρητή PCONP, ο χρονιστής 0, το ρολόι πραγματικού χρόνου (Real Time Clock) και ο ελεγκτής 1 του CAN. Η απόδοση μηδενικής τιμής στον καταχωρητή PINSEL1 έχει ως αποτέλεσμα να επιλέγεται η λειτουργία των ακροδεκτών που συνδέονται σε φυσικό επίπεδο με τον περιβάλλον του LPC2129 ως απλές ψηφιακές είσοδοι/έξοδοι. Οι μονάδες στον καταχωρητή IO0DIR ορίζουν ποιοι από τους ακροδέκτες λειτουργούν ως έξοδοι ενώ τα μηδενικά ως είσοδοι. Αρχικοποιείται το σύστημα χρονισμού επειδή στη συνέχεια γίνονται κλήσεις της συνάρτησης pause. Αρχικοποιείται επίσης ο ελεγκτής CAN. Ορίζεται η ένδειξη του 7-segment display σε τρεις παύλες έως ότου ο θάλαμος τοποθετηθεί στο ισόγειο και αρχικοποιούνται τα μηνύματα που πρόκειται να αποσταλλούν από τον κόμβο. Στη συνέχεια αποστέλλεται ένα μήνυμα που περιέχει την τρέχουσα κατάσταση του ανελκυστήρα και ξεκινάει ο αέναος βρόχος.

Η εκτέλεση του αέναου βρόχου ελέγχει την τρέχουσα κατάσταση του ανελκυστήρα για τυχόν αλλαγές και σε αυτήν την περίπτωση αποστέλλει το αντίστοιχο μήνυμα στο δίαυλο για την ενημέρωση του Κεντρικού Κόμβου. Με τον τρόπο αυτό και με δεδομένο ότι ο κύριος βρόχος εκτελείται σε πολύ τακτά χρονικά διαστήματα εξασφαλίζεται ότι ο Κεντρικός Κόμβος είναι πάντα ενήμερος για την πραγματική

κατάσταση του θαλάμου. Στην επόμενη εντολή του βρόχου ελέγχεται αν πιέστηκε το πλήκτρο παύσης κίνησης, οπότε και ο ανελκυστήρας σταματάει να κινείται. Ανάλογα με την τιμή της μεταβλητής isStopped, εκτελείται μία από τις δύο κύριες συναρτήσεις και στο τέλος κάθε εκτέλεσης του βρόχου ελέγχεται ο δίαυλος για ύπαρξη μηνύματος CAN.

Η παραπάνω υλοποίηση εξασφαλίζει ότι σε σύντομα και τακτά χρονικά διαστήματα ελέγχονται τόσο η κατάσταση του ανελκυστήρα όσο και αυτή του πλήκτρου STOP και του διαύλου CAN. Η εκτέλεση κάθε μιας από της κύριες συναρτήσεις δεν προκαλεί κάποια καθυστέρηση, με εξαίρεση την κατάσταση όπου ο θάλαμος απομακρύνεται από τον προηγούμενο όροφο στον οποίο βρισκόταν, η οποία όμως δεν διαρκεί τόσο ώστε να προκαλέσει δυσλειτουργία. Συνολικά, ο Κεντρικός Κόμβος έχει τον απόλυτο έλεγχο του ανελκυστήρα καθώς γνωρίζει την τρέχουσα κατάστασή του αφού ενημερώνεται σε κάθε της αλλαγή και οι εντολές του υπερισχύουν των όποιων εντολών εκτελεί ήδη ο Κόμβος Ανελκυστήρα.

ΜΕΤΑΦΟΡΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Η μεταφορά του αντίστοιχου προγράμματος σε κάθε κόμβο γίνεται με χρήση του συστήματος In-System Programming (ISP) και του εργαλείου LPC21ISP. Η ολοκληρωμένη πλακέτα συνδέεται διαμέσου της ενσωματωμένης σειριακής θύρας στη σειριακή θύρα COM1 του προσωπικού υπολογιστή στον οποίο βρίσκεται το δεκαεξαδικό αρχείο που έχει παράξει ο μεταφραστής. Για τη ρύθμιση των επιλογών εκτέλεσης του LPC21ISP απαιτείται η γνώση του τρόπου λειτουργίας του ελεγκτή UART όταν ο μικροεπεξεργαστής βρίσκεται σε κατάσταση ISP.

Για την είσοδο στην κατάσταση λειτουργίας ISP, το πρόγραμμα εκκίνησης (boot loader) που εκτελείται ύστερα από μία επανεκκίνηση του μικροεπεξεργαστή ελέγχει την κατάσταση του ακροδέκτη P0.14 και αν το βρει σε χαμηλό δυναμικό, αντί για την εκτέλεση του αποθηκευμένου προγράμματος του χρήστη, εκτελεί τη ρουτίνα εξυπηρέτησης του συστήματος ISP. Για τη σωστή επικοινωνία του μικροεπεξεργαστή με τον προσωπικό υπολογιστή, η σειριακή θύρα του δεύτερου πρέπει να είναι ρυθμισμένη σε 8 bit δεδομένων, 1 bit τερματισμού και κανένα bit ισοτιμίας. Η ταχύτητα μετάδοσης αναγνωρίζεται αυτόματα από τη ρουτίνα auto-baud, που εκτελείται πριν την κλήση των εντολών εξυπηρέτησης του ISP, ελέγχοντας τη λήψη συγκεκριμένης ακολουθίας χαρακτήρων. Αφού επιτευχθεί ο συχρονισμός, το πρόγραμμα που εκτελείται στον προσωπικό υπολογιστή αποστέλει τη συχνότητα του κρυστάλλου του μικροπεξεργαστή σε kHz και στη συνέχεια αναλαμβάνει να αποστείλει τις κατάλληλες εντολές ώστε να αποκτήσει πρόσβαση στη μνήμη flash του LPC2129 και να εγγράψει σε αυτήν το πρόγραμμα του χρήστη. Η επικοινωνία και ο έλεγχος του συστήματος πραγματοποιείται βάσει του προτωκόλλου που έχει ορίσει η κατασκευάστρια Philips με χρήση ορισμένων εντολών και μηνυμάτων και είναι διάφανη προς το χρήστη που εκτελεί το εργαλείο LPC21ISP.

Η μεταφορά των προγραμμάτων των δύο κόμβων φαίνεται στις παρακάτω εικόνες.

```
C:\CanProjects>lpc21isp_132x -control main.hex com1 38400 60000
lpc21isp version 1.32
File main.hex:
        loaded...
        converted to binary format...
        image size : 5060
Synchronizing..... OK
Read bootcode version: 1.65.0
Read part ID: LPC2194, 256 kiB ROM / 16 kiB SRAM (50462483)
Sector 0: ..........................................................................
..........................
Download Finished... taking 5 seconds
Now launching the brand new code
```

Εικόνα 40: Μεταφορά του προγράμματος του Κεντρικού Κόμβου

```
C:\CanProjects>lpc21isp_132x -control main.hex com1 38400 60000
lpc21isp version 1.32
File main.hex:
        loaded...
        converted to binary format...
        image size : 2764
Synchronizing. OK
Read bootcode version: 1.65.0
Read part ID: LPC2194, 256 kiB ROM / 16 kiB SRAM (50462483)
Sector 0: ..........................................................................
Download Finished... taking 4 seconds
Now launching the brand new code
```

Εικόνα 41: Μεταφορά του προγράμματος του Κόμβου Ανελκυστήρα

ΕΚΤΕΛΕΣΗ

Παρακάτω δίνονται οι εικόνες που βλέπει ο χρήστης του προσωπικού υπολογιστή σε δύο παραδείγματα εκτέλεσης της ολοκληρωμένης εφαρμογής. Και στα δύο παραδείγματα αυτά, ο Κεντρικός Κόμβος είναι συνδεδεμένος στη σειριακή θύρα COM1 του προσωπικού υπολογιστή, οι δύο Κόμβοι βρίσκονται συνδεδεμένοι στο δίαυλο CAN, ο Κόμβος Ανελκυστήρα είναι τοποθετημένος στο εσωτερικό του θαλάμου και συνδεδεμένος με τα πλήκτρα τοπικού ελέγχου και το ενδεικτικό 7-segment display, ενώ τέλος οι έξοδοι STEP και DIRECTION του Κόμβου Ανελκυστήρα τροφοδοτούν τις αντίστοιχες εισόδους του driver του βηματικού κινητήρα, ο οποίος έχει ρυθμιστεί σε αυτόματη λειτουργία, δηλαδή λαμβάνει τα σήματα ελέγχου του κινητήρα από τις εισόδους του και όχι από τα πλήκτρα που βρίσκονται στη διάτρητη πλακέτα του.

Η πρώτη εκτέλεση δίνει την παρακάτω εικόνα:

```
C:\CanProjects>lpc21isp_132x -termonly dummy.hex com1 38400 12000
lpc21isp version 1.32
Terminal started (press Escape to abort)


Welcome to the Central-node
The node accepts the inputs: m, h, s, e and u.
CAN successfully initialized!

Message -Elevator Status- received
Current Elevator Status:
Current Location: GND Floor
The elevator is empty
The elevator is stopped
The elevator alarm is OFF
Current Destination: GND Floor
No Command
No Command

Move To Floor: 2
CAN message -Move To Floor- successfully transmitted.

Message -Elevator Status- received
Current Elevator Status:
Current Location: Between GND and 1ST
The elevator is empty
The elevator is moving
The elevator alarm is OFF
Current Destination: 2ND Floor

Message -Elevator Status- received
Current Elevator Status:
Current Location: Between 1ST and 2ND
The elevator is empty
The elevator is moving
The elevator alarm is OFF
Current Destination: 2ND Floor

Message -Elevator Status- received
Current Elevator Status:
Current Location: 2ND Floor
The elevator is empty
The elevator is stopped
The elevator alarm is OFF
Current Destination: 2ND Floor
```

Εικόνα 42: Παράδειγμα 1 εκτέλεσης της εφαρμογής

Στο παραπάνω παράδειγμα, ο χρήστης αρχικά εκτελεί το εργαλείο LPC21ISP μόνο με λειτουργία τερματικού χρησιμοποιώντας τις κατάλληλες ρυθμίσεις, που είναι ίδιες με τις αντίστοιχες που έχουν οριστεί κατά την αρχικοποίηση του ελεγκτή UART στον κώδικα του Κεντρικού Κόμβου. Μόλις τυπωθεί το μήνυμα επιτυχούς αρχικοποίησης του κόμβου, λαμβάνεται το μήνυμα που περιέχει την κατάσταση του ανελκυστήρα και έχει σταλεί από το δεύτερο κόμβο οπότε τυπώνονται οι αντίστοιχες πληροφορίες και ο χρήστης πιέζει διαδοχικά δύο τυχαία πλήκτρα που δεν περιλαμβάνονται στο σύνολο των χαρακτήρων που αναγνωρίζει ο κόμβος, με αποτέλεσμα ο δεύτερος να απαντήσει με το μήνυμα «No Command».

Στη συνέχεια, ο χρήστης πιέζει το πλήκτρο «m» αποστέλλοντας τον αντίστοιχο χαρακτήρα. Ο Κόμβος απαντάει με το μήνυμα «Move To Floor:» επιβεβαιώνοντας τη

λήψη και αναγνώριση του χαρακτήρα και, αφού ο χρήστης πιέσει το χαρακτήρα «2», ενημερώνει για την επιτυχία της αποστολής του αντίστοιχου μηνύματος CAN στο δίαυλο. Ο Κόμβος Ανελκυστήρα έχει λάβει το μήνυμα και ο θάλαμος αρχίζει να μετακινείται προς τα επάνω, ώστε να προσεγγίσει το δεύτερο όροφο.

Με τη λήψη του μηνύματος, ο ανελκυστήρας αλλάζει προορισμό και από το ισόγειο που ήταν λόγω αρχικοποίησης μεταβάλλεται στο δεύτερο όροφο, σύμφωνα με την εντολή του χρήστη. Άμεσα ξεκινάει η μετακίνηση του θαλάμου και, καθώς ο ανελκυστήρας έχει αλλάξει κατάσταση, τόσο ως προς τον προορισμό όσο και ως προς τη θέση και την κατάσταση κίνησης, ο Κόμβος Ανελκυστήρα αποστέλλει το μήνυμα κατάστασης. Αυτό λαμβάνεται και τυπώνεται φορμαρισμένα από τον Κεντρικό Κόμβο. Όπως φαίνεται στην εικόνα παραπάνω, ο ανελκυστήρας πλέον βρίσκεται μεταξύ ισογείου και πρώτου ορόφου, κινείται και έχει προορισμό το δεύτερο όροφο.

Το επόμενο μήνυμα κατάστασης λαμβάνεται μόλις ο θάλαμος περάσει από τον πρώτο όροφο. Τότε, η κατάστασή του έχει αλλάξει μόνο ως προς τη θέση, η οποία είναι πλέον μεταξύ πρώτου και δευτέρου ορόφου, με τις υπόλοιπες μεταβλητές κατάστασης να παραμένουν ως είχαν. Ταυτόχρονα, η ένδειξη του 7-segment display που βρίσκεται στον ανελκυστήρα αλλάζει από 0 σε 1. Όταν ο θάλαμος φτάσει πράγματι στο δεύτερο όροφο, αποστέλλεται νέο μήνυμα κατάστασης, στο οποίο δηλώνεται ότι ανελκυστήρας βρίσκεται στο δεύτερο όροφο και είναι σταματημένος. Την ίδια στιγμή, η ένδειξη του 7-segment display μεταβάλλεται από 1 σε 2.

Η δεύτερη εκτέλεση δίνει την παρακάτω εικόνα:

```
Message -Elevator Status- received
Current Elevator Status:
Current Location: GND Floor
The elevator is full
The elevator is stopped
The elevator alarm is OFF
Current Destination: GND Floor

Message -Elevator Status- received
Current Elevator Status:
Current Location: Between GND and 1ST
The elevator is full
The elevator is moving
The elevator alarm is OFF
Current Destination: 2ND Floor

Stop
CAN message -Stop- successfully transmitted.

Message -Elevator Status- received
Current Elevator Status:
Current Location: Between GND and 1ST
The elevator is full
The elevator is stopped
The elevator alarm is OFF
Current Destination: 2ND Floor

Elevator call: GND Floor
CAN message -Move To Floor- successfully transmitted.

Message -Elevator Status- received
Current Elevator Status:
Current Location: Between GND and 1ST
The elevator is full
The elevator is moving
The elevator alarm is OFF
Current Destination: GND Floor

Message -Elevator Status- received
Current Elevator Status:
Current Location: GND Floor
The elevator is full
The elevator is stopped
The elevator alarm is OFF
Current Destination: GND Floor
```

Εικόνα 43: Παράδειγμα 2 εκτέλεσης της εφαρμογής

Στο δεύτερο παράδειγμα εκτέλεσης της εφαρμογής, ο ανελκυστήρας βρίσκεται αρχικά στο ισόγειο με τον αισθητήρα βάρους ενεργοποιημένο. Η κατάστασή του αυτή δηλώνεται στο πρώτο μήνυμα. Στη συνέχεια πιέζεται τοπικά το πλήκτρο του δευτέρου ορόφου και ο ανελκυστήρας αρχίζει να μετακινείται. Η μεταβολή αυτή δηλώνεται στο δεύτερο μήνυμα κατάστασης που λαμβάνει και τυπώνει ο Κεντρικός Κόμβος, όπου φαίνεται ότι ο θάλαμος κινείται μεταξύ ισογείου και πρώτου ορόφου με προορισμό το δεύτερο. Πριν όμως προλάβει να περάσει από τον πρώτο όροφο, γεγονός που θα προκαλούσε τη νέα αποστολή μηνύματος κατάστασης, ο χρήστης κεντρικά πιέζει το πλήκτρο «h» προκαλώντας την αναγνώριση της εντολής «STOP» (ή «HALT»). Ο Κεντρικός Κόμβος αποστέλλει το αντίστοιχο μήνυμα και ο ανελκυστήρας που το λαμβάνει σταματάει, αποστέλλοντας με τη σειρά του το νέο μήνυμα κατάστασης όπου δηλώνει ότι η θέση του

και ο προορισμός του δεν έχει μεταβληθεί αλλά είναι σταματημένος υπακούωντας στην εντολή του Κεντρικού Κόμβου.

Στο σημείο αυτό πιέζεται το πλήκτρο κλήσης ανελκυστήρα από το ισόγειο, πλήκτρο που ελέγχεται από τον Κεντρικό Κόμβο. Εφόσον ο ανελκυστήρας είναι σταματημένος, η κλήση του είναι δυνατή και επομένως ο Κεντρικός Κόμβος ενημερώνει για το πάτημα τυπώνοντας το μήνυμα «Elevator call: GND Floor» και αμέσως στέλνει στο δίαυλο CAN το μήνυμα που περιέχει την εντολή μετάβασης στο ισόγειο. Ο ανελκυστήρας αντιδρά κινούμενος προς το ισόγειο και ενημερώνει τον Κεντρικό Κόμβο με μήνυμα στο οποίο δηλώνεται ότι η θέση είναι μεν η ίδια αλλά πλέον ο θάλαμος κινείται και έχει τελικό προορισμό το ισόγειο. Μόλις φτάσει εκεί αποστέλλεται και το τελευταίο μήνυμα, όπου φαίνεται ότι ο ανελκυστήρας είναι πράγματι στο ισόγειο και σταματημένος.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1]. Philips Semiconductors (2003). *SOT314-2: LCP2129 Package Outline.*

[2]. Keil, An ARM Company (2008). *MCB2100 Evaluation board.* http://www.keil.com/mcb210/.

[3]. Embedded Artists AB (2005). *LPC2129 CAN Quickstart Board – User's Guide.*

[4]. Philips Semiconductors (2006). *LPC2119/LPC2129 – Product data sheet (Rev. 04)*

[5]. Philips Semiconductors (2004). *LPC2119/2129/2194/2292/2294 – User Manual*

[6]. Martin T. (2005). *The insider's guide to the Philips ARM7-based microcontrollers.* Coventry: Hitex (UK) Ltd.

[7]. Texas Instruments (2002). *TPS70251 Datasheet.*

[8]. STMicroelectronics (2003). *ST3232E Datasheet.*

[9]. Philips Semiconductors (2003). *TJA1041 Product Specification.*

[10]. Wikimedia Foundation (2008). *Stepper Motor.* http://en.wikipedia.org/wiki/Stepper_motor.

[11]. Kemp D. (2007). *Step Motor Basics*. http://www.hossmachine.info

[12]. Berger Lahr Mechatronic (2004). *Basic Products – 2-phase stepping motors*.

[13]. Embedded Artists AB (2005). *QuickStart Program Development – User's Guide*.

[14]. National Semiconductor (2006). *LM555 Timer Datasheed*.

ΠΑΡΑΡΤΗΜΑ Α

Στο Παράρτημα Α περιλαμβάνεται το σύνολο του πηγαίου κώδικα της εφαρμογής, ξεκινώντας από τα κοινά αρχεία για τους δύο κόμβους και συνεχίζοντας με τα αρχεία που ξεχωρίζουν τη λειτουργία κάθε κόμβου.

ΚΟΙΝΑ ΑΡΧΕΙΑ

types.h

```
/******************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module defines some regularly used typedefs
 *
 ******************************************************************************/

#ifndef INC_TYPES_H
#define INC_TYPES_H

/*  typedefs are here  */
typedef unsigned char       uint8_t;
typedef   signed char       int8_t;
typedef unsigned short      uint16_t;
typedef   signed short      int16_t;
typedef unsigned long       uint32_t;
typedef   signed long       int32_t;
typedef unsigned long long uint64_t;
typedef   signed long long  int64_t;

typedef enum {False, True}  boolean;

#endif
```

LPC21xx.h

```
/******************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC21xx ARM Processors
 * Copyright 2004 R O SoftWare
 * (extended for CAN - Copyright 2007 Alexandros Papanastasatos)
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 ******************************************************************************/
#ifndef INC_LPC21xx_H
#define INC_LPC21xx_H
```

```
#define REG_8          volatile unsigned char
#define REG16          volatile unsigned short
#define REG32          volatile unsigned long

#include "lpcWD.h"
#include "lpcTMR.h"
#include "lpcUART.h"
#include "lpcI2C.h"
#include "lpcSPI.h"
#include "lpcRTC.h"
#include "lpcGPIO.h"
#include "lpcPIN.h"
#include "lpcADC.h"
#include "lpcSCB.h"
#include "lpcVIC.h"
#include "lpcCAN.h"

//////////////////////////////////////////////////////////////////////////
// Watchdog
#define WD             ((wdRegs_t *)0xE0000000)

// Watchdog Registers
#define WDMOD          WD->mod        /* Watchdog Mode Register */
#define WDTC           WD->tc         /* Watchdog Time Constant Register */
#define WDFEED         WD->feed       /* Watchdog Feed Register */
#define WDTV           WD->tv         /* Watchdog Time Value Register */

//////////////////////////////////////////////////////////////////////////
// Timer 0
#define TMR0           ((pwmTmrRegs_t *)0xE0004000)

// Timer 0 Registers
#define T0IR           TMR0->ir       /* Interrupt Register */
#define T0TCR          TMR0->tcr      /* Timer Control Register */
#define T0TC           TMR0->tc       /* Timer Counter */
#define T0PR           TMR0->pr       /* Prescale Register */
#define T0PC           TMR0->pc       /* Prescale Counter Register */
#define T0MCR          TMR0->mcr      /* Match Control Register */
#define T0MR0          TMR0->mr0      /* Match Register 0 */
#define T0MR1          TMR0->mr1      /* Match Register 1 */
#define T0MR2          TMR0->mr2      /* Match Register 2 */
#define T0MR3          TMR0->mr3      /* Match Register 3 */
#define T0CCR          TMR0->ccr      /* Capture Control Register */
#define T0CR0          TMR0->cr0      /* Capture Register 0 */
#define T0CR1          TMR0->cr1      /* Capture Register 1 */
#define T0CR2          TMR0->cr2      /* Capture Register 2 */
#define T0CR3          TMR0->cr3      /* Capture Register 3 */
#define T0EMR          TMR0->emr      /* External Match Register */

//////////////////////////////////////////////////////////////////////////
// Timer 1
#define TMR1           ((pwmTmrRegs_t *)0xE0008000)

// Timer 1 Registers
#define T1IR           TMR1->ir       /* Interrupt Register */
#define T1TCR          TMR1->tcr      /* Timer Control Register */
#define T1TC           TMR1->tc       /* Timer Counter */
#define T1PR           TMR1->pr       /* Prescale Register */
#define T1PC           TMR1->pc       /* Prescale Counter Register */
#define T1MCR          TMR1->mcr      /* Match Control Register */
#define T1MR0          TMR1->mr0      /* Match Register 0 */
#define T1MR1          TMR1->mr1      /* Match Register 1 */
#define T1MR2          TMR1->mr2      /* Match Register 2 */
#define T1MR3          TMR1->mr3      /* Match Register 3 */
#define T1CCR          TMR1->ccr      /* Capture Control Register */
```

```
#define T1CR0          TMR1->cr0       /* Capture Register 0 */
#define T1CR1          TMR1->cr1       /* Capture Register 1 */
#define T1CR2          TMR1->cr2       /* Capture Register 2 */
#define T1CR3          TMR1->cr3       /* Capture Register 3 */
#define T1EMR          TMR1->emr       /* External Match Register */

////////////////////////////////////////////////////////////////////////////
// Pulse Width Modulator (PWM)
#define PWM            ((pwmTmrRegs_t *)0xE0014000)

// PWM Registers
#define PWMIR          PWM->ir         /* Interrupt Register */
#define PWMTCR         PWM->tcr        /* Timer Control Register */
#define PWMTC          PWM->tc         /* Timer Counter */
#define PWMPR          PWM->pr         /* Prescale Register */
#define PWMPC          PWM->pc         /* Prescale Counter Register */
#define PWMMCR         PWM->mcr        /* Match Control Register */
#define PWMMR0         PWM->mr0        /* Match Register 0 */
#define PWMMR1         PWM->mr1        /* Match Register 1 */
#define PWMMR2         PWM->mr2        /* Match Register 2 */
#define PWMMR3         PWM->mr3        /* Match Register 3 */
#define PWMMR4         PWM->mr4        /* Match Register 4 */
#define PWMMR5         PWM->mr5        /* Match Register 5 */
#define PWMMR6         PWM->mr6        /* Match Register 6 */
#define PWMPCR         PWM->pcr        /* Control Register */
#define PWMLER         PWM->ler        /* Latch Enable Register */

////////////////////////////////////////////////////////////////////////////
// Universal Asynchronous Receiver Transmitter 0 (UART0)
#define UART0          ((uartRegs_t *)0xE000C000)
#define U0_PINSEL      (0x00000005)    /* PINSEL0 Value for UART0 */
#define U0_PINMASK     (0x0000000F)    /* PINSEL0 Mask for UART0 */

// UART0 Registers
#define U0RBR          UART0->rbr      /* Receive Buffer Register */
#define U0THR          UART0->thr      /* Transmit Holding Register */
#define U0IER          UART0->ier      /* Interrupt Enable Register */
#define U0IIR          UART0->iir      /* Interrupt ID Register */
#define U0FCR          UART0->fcr      /* FIFO Control Register */
#define U0LCR          UART0->lcr      /* Line Control Register */
#define U0LSR          UART0->lsr      /* Line Status Register */
#define U0SCR          UART0->scr      /* Scratch Pad Register */
#define U0DLL          UART0->dll      /* Divisor Latch Register (LSB) */
#define U0DLM          UART0->dlm      /* Divisor Latch Register (MSB) */

////////////////////////////////////////////////////////////////////////////
// Universal Asynchronous Receiver Transmitter 1 (UART1)
#define UART1          ((uartRegs_t *)0xE0010000)
#define U1_PINSEL      (0x00050000)    /* PINSEL0 Value for UART1 */
#define U1_PINMASK     (0x000F0000)    /* PINSEL0 Mask for UART1 */

// UART1 Registers
#define U1RBR          UART1->rbr      /* Receive Buffer Register */
#define U1THR          UART1->thr      /* Transmit Holding Register */
#define U1IER          UART1->ier      /* Interrupt Enable Register */
#define U1IIR          UART1->iir      /* Interrupt ID Register */
#define U1FCR          UART1->fcr      /* FIFO Control Register */
#define U1LCR          UART1->lcr      /* Line Control Register */
#define U1MCR          UART1->mcr      /* MODEM Control Register */
#define U1LSR          UART1->lsr      /* Line Status Register */
#define U1MSR          UART1->msr      /* MODEM Status Register */
#define U1SCR          UART1->scr      /* Scratch Pad Register */
#define U1DLL          UART1->dll      /* Divisor Latch Register (LSB) */
#define U1DLM          UART1->dlm      /* Divisor Latch Register (MSB) */

////////////////////////////////////////////////////////////////////////////
```

```
// I2C Interface
#define I2C             ((i2cRegs_t *)0xE001C000)

// I2C Registers
#define I2CONSET        I2C->conset     /* Control Set Register */
#define I2STAT          I2C->stat       /* Status Register */
#define I2DAT           I2C->dat        /* Data Register */
#define I2ADR           I2C->adr        /* Slave Address Register */
#define I2SCLH          I2C->sclh       /* SCL Duty Cycle Register (high half word) */
#define I2SCLL          I2C->scll       /* SCL Duty Cycle Register (low half word) */
#define I2CONCLR        I2C->conclr     /* Control Clear Register */

////////////////////////////////////////////////////////////////////////////
// Serial Peripheral Interface 0 (SPI0)
#define SPI0            ((spiRegs_t *)0xE0020000)

// SPI0 Registers
#define S0SPCR          SPI0->cr        /* Control Register */
#define S0SPSR          SPI0->sr        /* Status Register */
#define S0SPDR          SPI0->dr        /* Data Register */
#define S0SPCCR         SPI0->ccr       /* Clock Counter Register */
#define S0SPINT         SPI0->flag      /* Interrupt Flag Register */

////////////////////////////////////////////////////////////////////////////
// Serial Peripheral Interface 1 (SPI1)
#define SPI1            ((spiRegs_t *)0xE0030000)

// SPI1 Registers
#define S1SPCR          SPI1->cr        /* Control Register */
#define S1SPSR          SPI1->sr        /* Status Register */
#define S1SPDR          SPI1->dr        /* Data Register */
#define S1SPCCR         SPI1->ccr       /* Clock Counter Register */
#define S1SPINT         SPI1->flag      /* Interrupt Flag Register */

////////////////////////////////////////////////////////////////////////////
// Real Time Clock
#define RTC             ((rtcRegs_t *)0xE0024000)

// RTC Registers
#define RTCILR          RTC->ilr        /* Interrupt Location Register */
#define RTCCTC          RTC->ctc        /* Clock Tick Counter */
#define RTCCCR          RTC->ccr        /* Clock Control Register */
#define RTCCIIR         RTC->ciir       /* Counter Increment Interrupt Register */
#define RTCAMR          RTC->amr        /* Alarm Mask Register */
#define RTCCTIME0       RTC->ctime0     /* Consolidated Time Register 0 */
#define RTCCTIME1       RTC->ctime1     /* Consolidated Time Register 1 */
#define RTCCTIME2       RTC->ctime2     /* Consolidated Time Register 2 */
#define RTCSEC          RTC->sec        /* Seconds Register */
#define RTCMIN          RTC->min        /* Minutes Register */
#define RTCHOUR         RTC->hour       /* Hours Register */
#define RTCDOM          RTC->dom        /* Day Of Month Register */
#define RTCDOW          RTC->dow        /* Day Of Week Register */
#define RTCDOY          RTC->doy        /* Day Of Year Register */
#define RTCMONTH        RTC->month      /* Months Register */
#define RTCYEAR         RTC->year       /* Years Register */
#define RTCALSEC        RTC->alsec      /* Alarm Seconds Register */
#define RTCALMIN        RTC->almin      /* Alarm Minutes Register */
#define RTCALHOUR       RTC->alhour     /* Alarm Hours Register */
#define RTCALDOM        RTC->aldom      /* Alarm Day Of Month Register */
#define RTCALDOW        RTC->aldow      /* Alarm Day Of Week Register */
#define RTCALDOY        RTC->aldoy      /* Alarm Day Of Year Register */
#define RTCALMON        RTC->almon      /* Alarm Months Register */
#define RTCALYEAR       RTC->alyear     /* Alarm Years Register */
#define RTCPREINT       RTC->preint     /* Prescale Value Register (integer) */
#define RTCPREFRAC      RTC->prefrac    /* Prescale Value Register (fraction) */
```

```
////////////////////////////////////////////////////////////////////////////
// General Purpose Input/Output
#define GPIO          ((gpioRegs_t *)0xE0028000)

// GPIO Registers
#define IO0PIN        GPIO->in0      /* P0 Pin Value Register */
#define IO0SET        GPIO->set0     /* P0 Pin Output Set Register */
#define IO0DIR        GPIO->dir0     /* P0 Pin Direction Register */
#define IO0CLR        GPIO->clr0     /* P0 Pin Output Clear Register */
#define IO1PIN        GPIO->in1      /* P1 Pin Value Register */
#define IO1SET        GPIO->set1     /* P1 Pin Output Set Register */
#define IO1DIR        GPIO->dir1     /* P1 Pin Direction Register */
#define IO1CLR        GPIO->clr1     /* P1 Pin Output Clear Register */

////////////////////////////////////////////////////////////////////////////
// Pin Connect Block
#define PINSEL        ((pinRegs_t *)0xE002C000)

// Pin Connect Block Registers
#define PINSEL0       PINSEL->sel0   /* Pin Function Select Register 0 */
#define PINSEL1       PINSEL->sel1   /* Pin Function Select Register 1 */
#define PINSEL2       PINSEL->sel2   /* Pin Function Select Register 2 */

////////////////////////////////////////////////////////////////////////////
// A/D Converter
#define ADC           ((adcRegs_t *)0xE0034000)

// A/D Converter Registers
#define ADCR          ADC->cr        /* Control Register */
#define ADDR          ADC->dr        /* Data Register */

////////////////////////////////////////////////////////////////////////////
// System Contol Block
#define SCB           ((scbRegs_t *)0xE01FC000)

// Memory Accelerator Module Registers (MAM)
#define MAMCR         SCB->mam.cr    /* Control Register */
#define MAMTIM        SCB->mam.tim   /* Timing Control Register */

// Memory Mapping Control Register
#define MEMMAP        SCB->memmap

// Phase Locked Loop Registers (PLL)
#define PLLCON        SCB->pll.con   /* Control Register */
#define PLLCFG        SCB->pll.cfg   /* Configuration Register */
#define PLLSTAT       SCB->pll.stat  /* Status Register */
#define PLLFEED       SCB->pll.feed  /* Feed Register */

// Power Control Registers
#define PCON          SCB->p.con     /* Control Register */
#define PCONP         SCB->p.conp    /* Peripherals Register */

// VPB Divider Register
#define VPBDIV        SCB->vpbdiv

// External Interrupt Registers
#define EXTINT        SCB->ext.flag  /* Flag Register */
#define EXTWAKE       SCB->ext.wake  /* Wakeup Register */
#define EXTMODE       SCB->ext.mode  /* Mode Register */
#define EXTPOLAR      SCB->ext.polar /* Polarity Register */

////////////////////////////////////////////////////////////////////////////
// Vectored Interrupt Controller
#define VIC           ((vicRegs_t *)0xFFFFF000)

// Vectored Interrupt Controller Registers
```

```
#define VICIRQStatus    VIC->irqStatus  /* IRQ Status Register */
#define VICFIQStatus    VIC->fiqStatus  /* FIQ Status Register */
#define VICRawIntr      VIC->rawIntr    /* Raw Interrupt Status Register */
#define VICIntSelect    VIC->intSelect  /* Interrupt Select Register */
#define VICIntEnable    VIC->intEnable  /* Interrupt Enable Register */
#define VICIntEnClear   VIC->intEnClear /* Interrupt Enable Clear Register */
#define VICSoftInt      VIC->softInt    /* Software Interrupt Register */
#define VICSoftIntClear VIC->softIntClear /* Software Interrupt Clear Register */
#define VICProtection   VIC->protection /* Protection Enable Register */
#define VICVectAddr     VIC->vectAddr   /* Vector Address Register */
#define VICDefVectAddr  VIC->defVectAddr /* Default Vector Address Register */
#define VICVectAddr0    VIC->vectAddr0  /* Vector Address 0 Register */
#define VICVectAddr1    VIC->vectAddr1  /* Vector Address 1 Register */
#define VICVectAddr2    VIC->vectAddr2  /* Vector Address 2 Register */
#define VICVectAddr3    VIC->vectAddr3  /* Vector Address 3 Register */
#define VICVectAddr4    VIC->vectAddr4  /* Vector Address 4 Register */
#define VICVectAddr5    VIC->vectAddr5  /* Vector Address 5 Register */
#define VICVectAddr6    VIC->vectAddr6  /* Vector Address 6 Register */
#define VICVectAddr7    VIC->vectAddr7  /* Vector Address 7 Register */
#define VICVectAddr8    VIC->vectAddr8  /* Vector Address 8 Register */
#define VICVectAddr9    VIC->vectAddr9  /* Vector Address 9 Register */
#define VICVectAddr10   VIC->vectAddr10 /* Vector Address 10 Register */
#define VICVectAddr11   VIC->vectAddr11 /* Vector Address 11 Register */
#define VICVectAddr12   VIC->vectAddr12 /* Vector Address 12 Register */
#define VICVectAddr13   VIC->vectAddr13 /* Vector Address 13 Register */
#define VICVectAddr14   VIC->vectAddr14 /* Vector Address 14 Register */
#define VICVectAddr15   VIC->vectAddr15 /* Vector Address 15 Register */
#define VICVectCntl0    VIC->vectCntl0  /* Vector Control 0 Register */
#define VICVectCntl1    VIC->vectCntl1  /* Vector Control 1 Register */
#define VICVectCntl2    VIC->vectCntl2  /* Vector Control 2 Register */
#define VICVectCntl3    VIC->vectCntl3  /* Vector Control 3 Register */
#define VICVectCntl4    VIC->vectCntl4  /* Vector Control 4 Register */
#define VICVectCntl5    VIC->vectCntl5  /* Vector Control 5 Register */
#define VICVectCntl6    VIC->vectCntl6  /* Vector Control 6 Register */
#define VICVectCntl7    VIC->vectCntl7  /* Vector Control 7 Register */
#define VICVectCntl8    VIC->vectCntl8  /* Vector Control 8 Register */
#define VICVectCntl9    VIC->vectCntl9  /* Vector Control 9 Register */
#define VICVectCntl10   VIC->vectCntl10 /* Vector Control 10 Register */
#define VICVectCntl11   VIC->vectCntl11 /* Vector Control 11 Register */
#define VICVectCntl12   VIC->vectCntl12 /* Vector Control 12 Register */
#define VICVectCntl13   VIC->vectCntl13 /* Vector Control 13 Register */
#define VICVectCntl14   VIC->vectCntl14 /* Vector Control 14 Register */
#define VICVectCntl15   VIC->vectCntl15 /* Vector Control 15 Register */

////////////////////////////////////////////////////////////////////////////
// CAN Controllers

// CAN Acceptance Filter RAM Base and End
#define CANAF_RAM               ((canAfRAM_t *)0xE0038000)

#define CANAF_RAM_START     CANAF_RAM->start        /* CAN AF RAM Start address */
#define CANAF_RAM_END       CANAF_RAM->end                  /* CAN AF RAM End address */

// CAN Acceptance Filter
#define CANAF                   ((canAfRegs_t *)0xE003C000)

// CAN Acceptance Filter Registers
#define AFMR                    CANAF->afmr                     /* Acceptance Filter Register */
#define SFF_sa                  CANAF->sff_sa           /* Standard Frame Individual Start Address
Register */
#define SFF_GRP_sa              CANAF->sff_grp_sa       /* Standard Frame Group Start Address
Register */
#define EFF_sa                  CANAF->eff_sa           /* Extended Frame Start Address Register */
#define EFF_GRP_sa              CANAF->eff_grp_sa       /* Extended Frame Group Start Address
Register */
#define ENDofTable              CANAF->endOfTable       /* End of AF Tables register */
```

```
#define LUTerrAd        CANAF->lutErrAd         /* LUT Error Address register */
#define LUTerr          CANAF->lutErr           /* LUT Error Register */

// CAN Central
#define CANCNTRL        ((canCntrlRegs_t *)0xE0040000)

// CAN Central Registers
#define CANTxSR                 CANCNTRL->canTxsr       /* CAN Central Transmit Status
Register */
#define CANRxSR                 CANCNTRL->canRxsr       /* CAN Central Receive Status
Register */
#define CANMSR          CANCNTRL->canmsr        /* CAN Central Miscellaneous Register */

// CAN Interface 1
#define CAN1            ((canRegs_t *)0xE0044000)

// CAN1 Registers
#define C1MOD           CAN1->mod               /* Mode Register */
#define C1CMR           CAN1->cmr               /* Command Register */
#define C1GSR           CAN1->gsr               /* Global Status Register */
#define C1ICR           CAN1->icr               /* Interrupt and Capture Register */
#define C1IER           CAN1->ier               /* Interrupt Enable Register */
#define C1BTR           CAN1->btr               /* Bus Timing Register */
#define C1EWL           CAN1->ewl               /* Error Warnign Limit Register */
#define C1SR            CAN1->sr                /* Status Register */
#define C1RFS           CAN1->rfs               /* Rx Frame Status Register */
#define C1RID           CAN1->rid               /* Rx Identifier Register */
#define C1RDA           CAN1->rda               /* Rx Data Register A */
#define C1RDB           CAN1->rdb               /* Rx Data Register B */
#define C1TFI1          CAN1->tfi1              /* Tx Frame Information Register (buffer 1) */
#define C1TID1          CAN1->tid1              /* Tx Identifier Register (buffer 1) */
#define C1TDA1          CAN1->tda1              /* Tx Data Register A (buffer 1) */
#define C1TDB1          CAN1->tdb1              /* Tx Data Register B (buffer 1) */
#define C1TFI2          CAN1->tfi2              /* Tx Frame Information Register (buffer 2) */
#define C1TID2          CAN1->tid2              /* Tx Identifier Register (buffer 2) */
#define C1TDA2          CAN1->tda2              /* Tx Data Register A (buffer 2) */
#define C1TDB2          CAN1->tdb2              /* Tx Data Register B (buffer 2) */
#define C1TFI3          CAN1->tfi3              /* Tx Frame Information Register (buffer 3) */
#define C1TID3          CAN1->tid3              /* Tx Identifier Register (buffer 3) */
#define C1TDA3          CAN1->tda3              /* Tx Data Register A (buffer 3) */
#define C1TDB3          CAN1->tdb3              /* Tx Data Register B (buffer 3) */

// CAN Interface 2
#define CAN2            ((canRegs_t *)0xE0048000)

// CAN2 Registers
#define C2MOD           CAN2->mod               /* Mode Register */
#define C2CMR           CAN2->cmr               /* Command Register */
#define C2GSR           CAN2->gsr               /* Global Status Register */
#define C2ICR           CAN2->icr               /* Interrupt and Capture Register */
#define C2IER           CAN2->ier               /* Interrupt Enable Register */
#define C2BTR           CAN2->btr               /* Bus Timing Register */
#define C2EWL           CAN2->ewl               /* Error Warnign Limit Register */
#define C2SR            CAN2->sr                /* Status Register */
#define C2RFS           CAN2->rfs               /* Rx Frame Status Register */
#define C2RID           CAN2->rid               /* Rx Identifier Register */
#define C2RDA           CAN2->rda               /* Rx Data Register A */
#define C2RDB           CAN2->rdb               /* Rx Data Register B */
#define C2TFI1          CAN2->tfi1              /* Tx Frame Information Register (buffer 1) */
#define C2TID1          CAN2->tid1              /* Tx Identifier Register (buffer 1) */
#define C2TDA1          CAN2->tda1              /* Tx Data Register A (buffer 1) */
#define C2TDB1          CAN2->tdb1              /* Tx Data Register B (buffer 1) */
#define C2TFI2          CAN2->tfi2              /* Tx Frame Information Register (buffer 2) */
#define C2TID2          CAN2->tid2              /* Tx Identifier Register (buffer 2) */
#define C2TDA2          CAN2->tda2              /* Tx Data Register A (buffer 2) */
#define C2TDB2          CAN2->tdb2              /* Tx Data Register B (buffer 2) */
```

```
#define C2TFI3                  CAN2->tfi3              /* Tx Frame Information Register (buffer 3) */
#define C2TID3                  CAN2->tid3              /* Tx Identifier Register (buffer 3) */
#define C2TDA3                  CAN2->tda3              /* Tx Data Register A (buffer 3) */
#define C2TDB3                  CAN2->tdb3              /* Tx Data Register B (buffer 3) */

// CAN Interface 3
#define CAN3                    ((canRegs_t *)0xE004C000)

// CAN3 Registers
#define C3MOD                   CAN3->mod               /* Mode Register */
#define C3CMR                   CAN3->cmr               /* Command Register */
#define C3GSR                   CAN3->gsr               /* Global Status Register */
#define C3ICR                   CAN3->icr               /* Interrupt and Capture Register */
#define C3IER                   CAN3->ier               /* Interrupt Enable Register */
#define C3BTR                   CAN3->btr               /* Bus Timing Register */
#define C3EWL                   CAN3->ewl               /* Error Warnign Limit Register */
#define C3SR                    CAN3->sr                /* Status Register */
#define C3RFS                   CAN3->rfs               /* Rx Frame Status Register */
#define C3RID                   CAN3->rid               /* Rx Identifier Register */
#define C3RDA                   CAN3->rda               /* Rx Data Register A */
#define C3RDB                   CAN3->rdb               /* Rx Data Register B */
#define C3TFI1                  CAN3->tfi1              /* Tx Frame Information Register (buffer 1) */
#define C3TID1                  CAN3->tid1              /* Tx Identifier Register (buffer 1) */
#define C3TDA1                  CAN3->tda1              /* Tx Data Register A (buffer 1) */
#define C3TDB1                  CAN3->tdb1              /* Tx Data Register B (buffer 1) */
#define C3TFI2                  CAN3->tfi2              /* Tx Frame Information Register (buffer 2) */
#define C3TID2                  CAN3->tid2              /* Tx Identifier Register (buffer 2) */
#define C3TDA2                  CAN3->tda2              /* Tx Data Register A (buffer 2) */
#define C3TDB2                  CAN3->tdb2              /* Tx Data Register B (buffer 2) */
#define C3TFI3                  CAN3->tfi3              /* Tx Frame Information Register (buffer 3) */
#define C3TID3                  CAN3->tid3              /* Tx Identifier Register (buffer 3) */
#define C3TDA3                  CAN3->tda3              /* Tx Data Register A (buffer 3) */
#define C3TDB3                  CAN3->tdb3              /* Tx Data Register B (buffer 3) */

// CAN Interface 4
#define CAN4                    ((canRegs_t *)0xE0050000)

// CAN4 Registers
#define C4MOD                   CAN4->mod               /* Mode Register */
#define C4CMR                   CAN4->cmr               /* Command Register */
#define C4GSR                   CAN4->gsr               /* Global Status Register */
#define C4ICR                   CAN4->icr               /* Interrupt and Capture Register */
#define C4IER                   CAN4->ier               /* Interrupt Enable Register */
#define C4BTR                   CAN4->btr               /* Bus Timing Register */
#define C4EWL                   CAN4->ewl               /* Error Warnign Limit Register */
#define C4SR                    CAN4->sr                /* Status Register */
#define C4RFS                   CAN4->rfs               /* Rx Frame Status Register */
#define C4RID                   CAN4->rid               /* Rx Identifier Register */
#define C4RDA                   CAN4->rda               /* Rx Data Register A */
#define C4RDB                   CAN4->rdb               /* Rx Data Register B */
#define C4TFI1                  CAN4->tfi1              /* Tx Frame Information Register (buffer 1) */
#define C4TID1                  CAN4->tid1              /* Tx Identifier Register (buffer 1) */
#define C4TDA1                  CAN4->tda1              /* Tx Data Register A (buffer 1) */
#define C4TDB1                  CAN4->tdb1              /* Tx Data Register B (buffer 1) */
#define C4TFI2                  CAN4->tfi2              /* Tx Frame Information Register (buffer 2) */
#define C4TID2                  CAN4->tid2              /* Tx Identifier Register (buffer 2) */
#define C4TDA2                  CAN4->tda2              /* Tx Data Register A (buffer 2) */
#define C4TDB2                  CAN4->tdb2              /* Tx Data Register B (buffer 2) */
#define C4TFI3                  CAN4->tfi3              /* Tx Frame Information Register (buffer 3) */
#define C4TID3                  CAN4->tid3              /* Tx Identifier Register (buffer 3) */
#define C4TDA3                  CAN4->tda3              /* Tx Data Register A (buffer 3) */
#define C4TDB3                  CAN4->tdb3              /* Tx Data Register B (buffer 3) */

#endif
```

lpcWD.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_WD_H
#define INC_LPC_WD_H

// Watchdog Registers
typedef struct
{
  REG_8 mod;                    // Watchdog Mode Register
  REG_8 _pad0[3];
  REG32 tc;                     // Watchdog Time Constant Register
  REG_8 feed;                   // Watchdog Feed Register
  REG32 tv;                     // Watchdog Time Value Register
} wdRegs_t;

#endif
```

lpcTMR.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_TMR_H
#define INC_LPC_TMR_H

// Timer & PWM Registers
typedef struct
{
  REG32 ir;                     // Interrupt Register
  REG32 tcr;                    // Timer Control Register
  REG32 tc;                     // Timer Counter
  REG32 pr;                     // Prescale Register
  REG32 pc;                     // Prescale Counter Register
  REG32 mcr;                    // Match Control Register
  REG32 mr0;                    // Match Register 0
  REG32 mr1;                    // Match Register 1
  REG32 mr2;                    // Match Register 2
  REG32 mr3;                    // Match Register 3
  REG32 ccr;                    // Capture Control Register
  REG32 cr0;                    // Capture Register 0
  REG32 cr1;                    // Capture Register 1
```

```
  REG32 cr2;                    // Capture Register 2
  REG32 cr3;                    // Capture Register 3
  REG32 emr;                    // External Match Register
  REG32 mr4;                    // Match Register 4
  REG32 mr5;                    // Match Register 5
  REG32 mr6;                    // Match Register 6
  REG32 pcr;                    // Control Register
  REG32 ler;                    // Latch Enable Register
} pwmTmrRegs_t;

// Timer Interrupt Register Bit Definitions
#define TIR_MR0I    (1 << 0)        // Interrupt flag for match channel 0
#define TIR_MR1I    (1 << 1)        // Interrupt flag for match channel 1
#define TIR_MR2I    (1 << 2)        // Interrupt flag for match channel 2
#define TIR_MR3I    (1 << 3)        // Interrupt flag for match channel 3
#define TIR_CR0I    (1 << 4)        // Interrupt flag for capture channel 0 event
#define TIR_CR1I    (1 << 5)        // Interrupt flag for capture channel 1 event
#define TIR_CR2I    (1 << 6)        // Interrupt flag for capture channel 2 event
#define TIR_CR3I    (1 << 7)        // Interrupt flag for capture channel 3 event

// PWM Interrupt Register Bit Definitions
#define PWMIR_MR0I  (1 << 0)        // Interrupt flag for match channel 0
#define PWMIR_MR1I  (1 << 1)        // Interrupt flag for match channel 1
#define PWMIR_MR2I  (1 << 2)        // Interrupt flag for match channel 2
#define PWMIR_MR3I  (1 << 3)        // Interrupt flag for match channel 3
#define PWMIR_MR4I  (1 << 8)        // Interrupt flag for match channel 4
#define PWMIR_MR5I  (1 << 9)        // Interrupt flag for match channel 5
#define PWMIR_MR6I  (1 << 10)       // Interrupt flag for match channel 6
#define PWMIR_MASK  (0x070F)

// Timer Control Register Bit Definitions
#define TCR_ENABLE  (1 << 0)
#define TCR_RESET   (1 << 1)

// PWM Control Register Bit Definitions
#define PWMCR_ENABLE (1 << 0)
#define PWMCR_RESET (1 << 1)

// Timer Match Control Register Bit Definitions
#define TMCR_MR0_I  (1 << 0)        // Enable Interrupt when MR0 matches TC
#define TMCR_MR0_R  (1 << 1)        // Enable Reset of TC upon MR0 match
#define TMCR_MR0_S  (1 << 2)        // Enable Stop of TC upon MR0 match
#define TMCR_MR1_I  (1 << 3)        // Enable Interrupt when MR1 matches TC
#define TMCR_MR1_R  (1 << 4)        // Enable Reset of TC upon MR1 match
#define TMCR_MR1_S  (1 << 5)        // Enable Stop of TC upon MR1 match
#define TMCR_MR2_I  (1 << 6)        // Enable Interrupt when MR2 matches TC
#define TMCR_MR2_R  (1 << 7)        // Enable Reset of TC upon MR2 match
#define TMCR_MR2_S  (1 << 8)        // Enable Stop of TC upon MR2 match
#define TMCR_MR3_I  (1 << 9)        // Enable Interrupt when MR3 matches TC
#define TMCR_MR3_R  (1 << 10)       // Enable Reset of TC upon MR3 match
#define TMCR_MR3_S  (1 << 11)       // Enable Stop of TC upon MR3 match

// Timer Capture Control Register Bit Definitions
#define TCCR_CR0_R (1 << 0)         // Enable Rising edge on CAPn.0 will load TC to CR0
#define TCCR_CR0_F (1 << 1)         // Enable Falling edge on CAPn.0 will load TC to CR0
#define TCCR_CR0_I (1 << 2)         // Enable Interrupt on load of CR0
#define TCCR_CR1_R (1 << 3)         // Enable Rising edge on CAPn.1 will load TC to CR1
#define TCCR_CR1_F (1 << 4)         // Enable Falling edge on CAPn.1 will load TC to CR1
#define TCCR_CR1_I (1 << 5)         // Enable Interrupt on load of CR1
#define TCCR_CR2_R (1 << 6)         // Enable Rising edge on CAPn.2 will load TC to CR2
#define TCCR_CR2_F (1 << 7)         // Enable Falling edge on CAPn.2 will load TC to CR2
#define TCCR_CR2_I (1 << 8)         // Enable Interrupt on load of CR2
#define TCCR_CR3_R (1 << 9)         // Enable Rising edge on CAPn.3 will load TC to CR3
#define TCCR_CR3_F (1 << 10)        // Enable Falling edge on CAPn.3 will load TC to CR3
#define TCCR_CR3_I (1 << 11)        // Enable Interrupt on load of CR3
```

```
#endif
```

lpcUART.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_UART_H
#define INC_LPC_UART_H

// Universal Asynchronous Receiver Transmitter Registers
typedef struct
{
  union
    {
    REG_8 rbr;                    // Receive Buffer Register
    REG_8 thr;                    // Transmit Holding Register
    REG_8 dll;                    // Divisor Latch Register (LSB)
    REG_8 _pad0[4];
    };

  union
    {
    REG_8 ier;                    // Interrupt Enable Register
    REG_8 dlm;                    // Divisor Latch Register (MSB)
    REG_8 _pad1[4];
    };

  union
    {
    REG_8 iir;                    // Interrupt ID Register
    REG_8 fcr;                    // FIFO Control Register
    REG_8 _pad2[4];
    };

  REG_8 lcr;                      // Line Control Registe
  REG_8 _pad3[3];
  REG_8 mcr;                      // MODEM Control Register
  REG_8 _pad4[3];
  REG_8 lsr;                      // Line Status Register
  REG_8 _pad5[3];
  REG_8 msr;                      // MODEM Status Register
  REG_8 _pad6[3];
  REG_8 scr;                      // Scratch Pad Register
  REG_8 _pad7[3];
} uartRegs_t;

////////////////////////////////////////////////////////////////////////////
// UART defines

// Interrupt Enable Register bit definitions
#define UIER_ERBFI        (1 << 0)    // Enable Receive Data Available Interrupt
#define UIER_ETBEI        (1 << 1)    // Enable Transmit Holding Register Empty Interrupt
#define UIER_ELSI         (1 << 2)    // Enable Receive Line Status Interrupt
#define UIER_EDSSI        (1 << 3)    // Enable MODEM Status Interrupt
```

```
// Interrupt ID Register bit definitions
#define UIIR_NO_INT        (1 << 0)   // NO INTERRUPTS PENDING
#define UIIR_MS_INT        (0 << 1)   // MODEM Status
#define UIIR_THRE_INT      (1 << 1)   // Transmit Holding Register Empty
#define UIIR_RDA_INT       (2 << 1)   // Receive Data Available
#define UIIR_RLS_INT       (3 << 1)   // Receive Line Status
#define UIIR_CTI_INT       (6 << 1)   // Character Timeout Indicator
#define UIIR_ID_MASK       0x0E

// FIFO Control Register bit definitions
#define UFCR_FIFO_ENABLE    (1 << 0)   // FIFO Enable
#define UFCR_RX_FIFO_RESET  (1 << 1)   // Reset Receive FIFO
#define UFCR_TX_FIFO_RESET  (1 << 2)   // Reset Transmit FIFO
#define UFCR_FIFO_TRIG1     (0 << 6)   // Trigger @ 1 character in FIFO
#define UFCR_FIFO_TRIG4     (1 << 6)   // Trigger @ 4 characters in FIFO
#define UFCR_FIFO_TRIG8     (2 << 6)   // Trigger @ 8 characters in FIFO
#define UFCR_FIFO_TRIG14    (3 << 6)   // Trigger @ 14 characters in FIFO

// Line Control Register bit definitions
#define ULCR_CHAR_5        (0 << 0)   // 5-bit character length
#define ULCR_CHAR_6        (1 << 0)   // 6-bit character length
#define ULCR_CHAR_7        (2 << 0)   // 7-bit character length
#define ULCR_CHAR_8        (3 << 0)   // 8-bit character length
#define ULCR_STOP_1        (0 << 2)   // 1 stop bit
#define ULCR_STOP_2        (1 << 2)   // 2 stop bits
#define ULCR_PAR_NO        (0 << 3)   // No Parity
#define ULCR_PAR_ODD       (1 << 3)   // Odd Parity
#define ULCR_PAR_EVEN      (3 << 3)   // Even Parity
#define ULCR_PAR_MARK      (5 << 3)   // MARK "1" Parity
#define ULCR_PAR_SPACE     (7 << 3)   // SPACE "0" Paruty
#define ULCR_BREAK_ENABLE   (1 << 6)   // Output BREAK line condition
#define ULCR_DLAB_ENABLE    (1 << 7)   // Enable Divisor Latch Access

// Modem Control Register bit definitions
#define UMCR_DTR           (1 << 0)   // Data Terminal Ready
#define UMCR_RTS           (1 << 1)   // Request To Send
#define UMCR_LB            (1 << 4)   // Loopback

// Line Status Register bit definitions
#define ULSR_RDR           (1 << 0)   // Receive Data Ready
#define ULSR_OE            (1 << 1)   // Overrun Error
#define ULSR_PE            (1 << 2)   // Parity Error
#define ULSR_FE            (1 << 3)   // Framing Error
#define ULSR_BI            (1 << 4)   // Break Interrupt
#define ULSR_THRE          (1 << 5)   // Transmit Holding Register Empty
#define ULSR_TEMT          (1 << 6)   // Transmitter Empty
#define ULSR_RXFE          (1 << 7)   // Error in Receive FIFO
#define ULSR_ERR_MASK      0x1E

// Modem Status Register bit definitions
#define UMSR_DCTS          (1 << 0)   // Delta Clear To Send
#define UMSR_DDSR          (1 << 1)   // Delta Data Set Ready
#define UMSR_TERI          (1 << 2)   // Trailing Edge Ring Indicator
#define UMSR_DDCD          (1 << 3)   // Delta Data Carrier Detect
#define UMSR_CTS           (1 << 4)   // Clear To Send
#define UMSR_DSR           (1 << 5)   // Data Set Ready
#define UMSR_RI            (1 << 6)   // Ring Indicator
#define UMSR_DCD           (1 << 7)   // Data Carrier Detect

#endif
```

lpcI2C.h

```
/*****************************************************************************
```

```
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_I2C_H
#define INC_LPC_I2C_H

// I2C Interface Registers
typedef struct
{
  REG_8 conset;                    // Control Set Register
  REG_8 _pad0[3];
  REG_8 stat;                      // Status Register
  REG_8 _pad1[3];
  REG_8 dat;                       // Data Register
  REG_8 _pad2[3];
  REG_8 adr;                       // Slave Address Register
  REG_8 _pad3[3];
  REG16 sclh;                      // SCL Duty Cycle Register (high half word)
  REG16 _pad4;
  REG16 scll;                      // SCL Duty Cycle Register (low half word)
  REG16 _pad5;
  REG_8 conclr;                    // Control Clear Register
  REG_8 _pad6[3];
} i2cRegs_t;

#endif
```

lpcSPI.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_SPI_H
#define INC_LPC_SPI_H

// Serial Peripheral Interface Registers (SPI)
typedef struct
{
  REG_8 cr;                        // Control Register
  REG_8 _pad0[3];
  REG_8 sr;                        // Status Register
  REG_8 _pad1[3];
  REG_8 dr;                        // Data Register
  REG_8 _pad2[3];
  REG_8 ccr;                       // Clock Counter Register
  REG_8 _pad3[3];
```

```
  REG_8 tcr;                    // Test Control Register
  REG_8 _pad4[3];
  REG_8 tsr;                    // Test Status Register
  REG_8 _pad5[3];
  REG_8 tor;                    // Test Observe Register
  REG_8 _pad6[3];
  REG_8 flag;                   // Interrupt Flag Register
  REG_8 _pad7[3];
} spiRegs_t;

#endif
```

lpcRTC.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_RTC_H
#define INC_LPC_RTC_H

typedef struct
{
  REG_8 ilr;                   // Interrupt Location Register
  REG_8 _pad0[3];
  REG16 ctc;                    // Clock Tick Counter
  REG16 _pad1;
  REG_8 ccr;                    // Clock Control Register
  REG_8 _pad2[3];
  REG_8 ciir;                  // Counter Increment Interrupt Register
  REG_8 _pad3[3];
  REG_8 amr;                    // Alarm Mask Register
  REG_8 _pad4[3];
  REG32 ctime0;                   // Consolidated Time Register 0
  REG32 ctime1;                   // Consolidated Time Register 1
  REG32 ctime2;                   // Consolidated Time Register 2
  REG_8 sec;                   // Seconds Register
  REG_8 _pad5[3];
  REG_8 min;                   // Minutes Register
  REG_8 _pad6[3];
  REG_8 hour;                   // Hours Register
  REG_8 _pad7[3];
  REG_8 dom;                    // Day Of Month Register
  REG_8 _pad8[3];
  REG_8 dow;                    // Day Of Week Register
  REG_8 _pad9[3];
  REG16 doy;                   // Day Of Year Register
  REG16 _pad10;
  REG_8 month;                    // Months Register
  REG_8 _pad11[3];
  REG16 year;                   // Years Register
  REG32 _pad12[8];
  REG_8 alsec;                   // Alarm Seconds Register
  REG_8 _pad13[3];
  REG_8 almin;                   // Alarm Minutes Register
  REG_8 _pad14[3];
```

```
  REG_8 alhour;                 // Alarm Hours Register
  REG_8 _pad15[3];
  REG_8 aldom;                  // Alarm Day Of Month Register
  REG_8 _pad16[3];
  REG_8 aldow;                  // Alarm Day Of Week Register
  REG_8 _pad17[3];
  REG16 aldoy;                  // Alarm Day Of Year Register
  REG16 _pad18;
  REG_8 almon;                  // Alarm Months Register
  REG_8 _pad19[3];
  REG16 alyear;                 // Alarm Years Register
  REG16 _pad20;
  REG16 preint;                 // Prescale Value Register (integer)
  REG16 _pad21;
  REG16 prefrac;                // Prescale Value Register (fraction)
  REG16 _pad22;
} rtcRegs_t;

#endif
```

lpcGPIO.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_GPIO_H
#define INC_LPC_GPIO_H

// General Purpose Input/Output Registers (GPIO)
typedef struct
{
  REG32 in0;                    // P0 Pin Value Register
  REG32 set0;                   // P0 Pin Output Set Register
  REG32 dir0;                   // P0 Pin Direction Register
  REG32 clr0;                   // P0 Pin Output Clear Register
  REG32 in1;                    // P1 Pin Value Register
  REG32 set1;                   // P1 Pin Output Set Register
  REG32 dir1;                   // P1 Pin Direction Register
  REG32 clr1;                   // P1 Pin Output Clear Register
  REG32 in2;                    // P2 Pin Value Register
  REG32 set2;                   // P2 Pin Output Set Register
  REG32 dir2;                   // P2 Pin Direction Register
  REG32 clr2;                   // P2 Pin Output Clear Register
  REG32 in3;                    // P3 Pin Value Register
  REG32 set3;                   // P3 Pin Output Set Register
  REG32 dir3;                   // P3 Pin Direction Register
  REG32 clr3;                   // P3 Pin Output Clear Register
} gpioRegs_t;

#endif
```

lpcPIN.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_PIN_H
#define INC_LPC_PIN_H

// Pin Connect Block Registers
typedef struct
{
  REG32 sel0;                   // Pin Function Select Register 0
  REG32 sel1;                   // Pin Function Select Register 1
  REG32 _pad[3];
  REG32 sel2;                   // Pin Function Select Register 2
} pinRegs_t;

#endif
```

lpcADC.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_ADC_H
#define INC_LPC_ADC_H

// A/D Converter Registers
typedef struct
{
  REG32 cr;                     // Control Register
  REG32 dr;                     // Data Register
} adcRegs_t;

#endif
```

lpcSCB.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
```

```
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_SCB_H
#define INC_LPC_SCB_H

// System Control Block Registers
typedef struct
{
  // Memory Accelerator Module Registers (MAM)
  struct
    {
    REG_8 cr;                       // Control Register
    REG_8 _pad0[3];
    REG_8 tim;                      // Timing Control Register
    REG32 _pad1[14];
    } mam;

  // Memory Mapping Control Register
  REG_8 memmap;
  REG32 _pad0[15];

  // Phase Locked Loop Registers (PLL)
  struct
    {
    REG_8 con;                      // Control Register
    REG_8 _pad0[3];
    REG_8 cfg;                      // Configuration Register
    REG_8 _pad1[3];
    REG16 stat;                     // Status Register
    REG16 _pad2;
    REG_8 feed;                     // Feed Register
    REG32 _pad3[12];
    } pll;

  // Power Control Registers
  struct
    {
    REG_8 con;                      // Control Register
    REG_8 _pad0[3];
    REG32 conp;                     // Peripherals Register
    REG32 _pad1[14];
    } p;

  // VPB Divider Register
  REG_8 vpbdiv;
  REG32 _pad1[15];

  // External Interrupt Registers
  struct
    {
    REG_8 flag;                     // Flag Register
    REG_8 _pad0[3];
    REG_8 wake;                     // Wakeup Register
    REG_8 _pad1[3];
    REG_8 mode;                     // Mode Register
    REG_8 _pad2[3];
    REG_8 polar;                    // Polarity Register
    REG32 _pad3[12];
    } ext;
```

```
} scbRegs_t;


////////////////////////////////////////////////////////////////////////////
// MAM defines
#define MAMCR_OFF     0
#define MAMCR_PART    1
#define MAMCR_FULL    2

#define MAMTIM_CYCLES (((CCLK) + 19999999) / 20000000)

////////////////////////////////////////////////////////////////////////////
// MEMMAP defines
#define MEMMAP_BBLK   0               // Interrupt Vectors in Boot Block
#define MEMMAP_FLASH  1               // Interrupt Vectors in Flash
#define MEMMAP_SRAM   2               // Interrupt Vectors in SRAM

////////////////////////////////////////////////////////////////////////////
// PLL defines & computations
// Compute the value of PLL_DIV and test range validity
// FOSC & PLL_MUL should be defined in project configuration file (config.h)
#ifndef CCLK
#define CCLK          (FOSC * PLL_MUL)  // CPU Clock Freq.
#endif

#define FCCO_MAX      (320000000)       // Max CC Osc Freq.
#define PLL_DIV       (FCCO_MAX / (2 * CCLK)) // PLL Divider
#define FCCO          (FOSC * PLL_MUL * 2 * PLL_DIV) // CC Osc. Freq.

// PLLCON Register Bit Definitions
#define PLLCON_PLLE   (1 << 0)          // PLL Enable
#define PLLCON_PLLC   (1 << 1)          // PLL Connect

// PLLCFG Register Bit Definitions
#define PLLCFG_MSEL   ((PLL_MUL - 1) << 0) // PLL Multiplier
#define PLLCFG_PSEL   ((PLL_DIV - 1) << 5) // PLL Divider

// PLLSTAT Register Bit Definitions
#define PLLSTAT_LOCK  (1 << 10)         // PLL Lock Status Bit

////////////////////////////////////////////////////////////////////////////
// VPBDIV defines & computations
#define VPBDIV_VALUE  (PBSD & 0x03)     // VPBDIV value

#endif
```

lpcVIC.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2004 R O SoftWare
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_VIC_H
#define INC_LPC_VIC_H

// Vectored Interrupt Controller Registers (VIC)
```

```
typedef struct
{
  REG32 irqStatus;                  // IRQ Status Register
  REG32 fiqStatus;                  // FIQ Status Register
  REG32 rawIntr;                    // Raw Interrupt Status Register
  REG32 intSelect;                  // Interrupt Select Register
  REG32 intEnable;                  // Interrupt Enable Register
  REG32 intEnClear;                 // Interrupt Enable Clear Register
  REG32 softInt;                    // Software Interrupt Register
  REG32 softIntClear;               // Software Interrupt Clear Register
  REG32 protection;                 // Protection Enable Register
  REG32 _pad0[3];
  REG32 vectAddr;                   // Vector Address Register
  REG32 defVectAddr;                // Default Vector Address Register
  REG32 _pad1[50];
  REG32 vectAddr0;                  // Vector Address 0 Register
  REG32 vectAddr1;                  // Vector Address 1 Register
  REG32 vectAddr2;                  // Vector Address 2 Register
  REG32 vectAddr3;                  // Vector Address 3 Register
  REG32 vectAddr4;                  // Vector Address 4 Register
  REG32 vectAddr5;                  // Vector Address 5 Register
  REG32 vectAddr6;                  // Vector Address 6 Register
  REG32 vectAddr7;                  // Vector Address 7 Register
  REG32 vectAddr8;                  // Vector Address 8 Register
  REG32 vectAddr9;                  // Vector Address 9 Register
  REG32 vectAddr10;                 // Vector Address 10 Register
  REG32 vectAddr11;                 // Vector Address 11 Register
  REG32 vectAddr12;                 // Vector Address 12 Register
  REG32 vectAddr13;                 // Vector Address 13 Register
  REG32 vectAddr14;                 // Vector Address 14 Register
  REG32 vectAddr15;                 // Vector Address 15 Register
  REG32 _pad2[48];
  REG32 vectCntl0;                  // Vector Control 0 Register
  REG32 vectCntl1;                  // Vector Control 1 Register
  REG32 vectCntl2;                  // Vector Control 2 Register
  REG32 vectCntl3;                  // Vector Control 3 Register
  REG32 vectCntl4;                  // Vector Control 4 Register
  REG32 vectCntl5;                  // Vector Control 5 Register
  REG32 vectCntl6;                  // Vector Control 6 Register
  REG32 vectCntl7;                  // Vector Control 7 Register
  REG32 vectCntl8;                  // Vector Control 8 Register
  REG32 vectCntl9;                  // Vector Control 9 Register
  REG32 vectCntl10;                 // Vector Control 10 Register
  REG32 vectCntl11;                 // Vector Control 11 Register
  REG32 vectCntl12;                 // Vector Control 12 Register
  REG32 vectCntl13;                 // Vector Control 13 Register
  REG32 vectCntl14;                 // Vector Control 14 Register
  REG32 vectCntl15;                 // Vector Control 15 Register
} vicRegs_t;

// VIC Channel Assignments
#define VIC_WDT         0
#define VIC_TIMER0      4
#define VIC_TIMER1      5
#define VIC_UART0       6
#define VIC_UART1       7
#define VIC_PWM         8
#define VIC_PWM0        8
#define VIC_I2C         9
#define VIC_SPI         10
#define VIC_SPI0        10
#define VIC_SPI1        11
#define VIC_PLL         12
#define VIC_RTC         13
#define VIC_EINT0       14
#define VIC_EINT1       15
```

```
#define VIC_EINT2       16
#define VIC_EINT3       17
#define VIC_ADC         18

// Vector Control Register bit definitions
#define VIC_ENABLE      (1 << 5)

// Convert Channel Number to Bit Value
#define VIC_BIT(chan)   (1 << (chan))

#endif
```

lpcCAN.h

```
/******************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * Header file for Philips LPC ARM Processors.
 * Copyright 2007 Alexandros Papanastasatos
 * (based on Header files by R O Software)
 *
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_LPC_CAN_H
#define INC_LPC_CAN_H

// CAN Controllers Registers

// CAN Acceptance Filter RAM
typedef struct
{
  REG32 start;                  // CAN AF RAM Start address
  REG32 _pad0[510];
  REG32 end;                    // CAN AF RAM End address
} canAfRAM_t;

// CAN Acceptance Filter Registers
typedef struct
{
  REG32 afmr;                   // Acceptance Filter Register
  REG32 sff_sa;                 // Standard Frame Individual Start Address Register
  REG32 sff_grp_sa;             // Standard Frame Group Start Address Register
  REG32 eff_sa;                 // Extended Frame Start Address Register
  REG32 eff_grp_sa;             // Extended Frame Group Start Address Register
  REG32 endOfTable;             // End of AF Tables register
  REG32 lutErrAd;               // LUT Error Address register
  REG32 lutErr;    // LUT Error Register
} canAfRegs_t;

// Central CAN Registers
typedef struct
{
  REG32 canTxsr;                      // CAN Central Transmit Status Register
  REG32 canRxsr;                      // CAN Central Receive Status Register
  REG32 canmsr;                     // CAN Central Miscellaneous Register
} canCntrlRegs_t;

// CAN Interface Registers
typedef struct
{
```

```
  REG32 mod;                // Controls the operating mode of the CAN Controller
  REG32 cmr;                // Command bits that affect the state of the CAN Controller
  REG32 gsr;                // Global Controller Status and Error Counters
  REG32 icr;                // Interrupt status, Arbitration Lost Capture, Error Code Capture
  REG32 ier;                // Interrupt Enable
  REG32 btr;                // Bus Timing
  REG32 ewl;                // Error Warning Limit
  REG32 sr;                 // Status Register
  REG32 rfs;                // Receive frame status
  REG32 rid;                // Received Identifier
  REG32 rda;                // Received data bytes 1-4
  REG32 rdb;                // Received data bytes 5-8
  REG32 tfi1;               // Transmit frame info (1)
  REG32 tid1;               // Transmit Identifier (1)
  REG32 tda1;               // Transmit data bytes 1-4 (1)
  REG32 tdb1;               // Transmit data bytes 5-8 (1)
  REG32 tfi2;               // Transmit frame info (2)
  REG32 tid2;               // Transmit Identifier (2)
  REG32 tda2;               // Transmit data bytes 1-4 (2)
  REG32 tdb2;               // Transmit data bytes 5-8 (2)
  REG32 tfi3;               // Transmit frame info (3)
  REG32 tid3;               // Transmit Identifier (3)
  REG32 tda3;               // Transmit data bytes 1-4 (3)
  REG32 tdb3;               // Transmit data bytes 5-8 (3)
} canRegs_t;

#endif
```

config.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides information about the project configuration
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_CONFIG_H
#define INC_CONFIG_H

#include "types.h"
#include "LPC21xx.h"

// some handy DEFINES
#ifndef FALSE
#define FALSE           0
#ifndef TRUE
#define TRUE            !FALSE
#endif
#endif

#ifndef BIT
#define BIT(n)          (1L << (n))
#endif

// declare functions and values from crt0.S & the linker control file
extern void reset(void);
// extern void exit(void);
extern void abort(void);
// maybe add interrupt vector addresses
```

```
#define HOST_BAUD          (38400)

#define WDOG()

// PLL setup values are computed within the LPC include file
// It relies upon the following defines
#define FOSC               (12000000)  // Master Oscillator Freq.
#define PLL_MUL              (5)       // PLL Multiplier
#define CCLK               (FOSC * PLL_MUL) // CPU Clock Freq.

// Pheripheral Bus Speed Divider
#define PBSD               2         // MUST BE 1, 2, or 4
#define PCLK               (CCLK / PBSD) // Pheripheal Bus Clock Freq.

// Do some value range testing
#if ((FOSC < 10000000) || (FOSC > 25000000))
#error Fosc out of range (10MHz-25MHz)
#error correct and recompile
#endif

#if ((CCLK < 10000000) || (CCLK > 60000000))
#error cclk out of range (10MHz-60MHz)
#error correct PLL_MUL and recompile
#endif

#if ((FCCO < 150000000) || (FCCO > 320000000))
#error Fcco out of range (156MHz-320MHz)
#error internal algorithm error
#endif

#if ((PBSD != 1) && (PBSD != 2) && (PBSD != 4))
#error Pheripheal Bus Speed Divider (PBSD) illegal value (1, 2, or 4)
#endif

// Port Bit Definitions & Macros:    Description - initial conditions
#define TXD0_BIT           BIT(0)      // used by UART0
#define RXD0_BIT           BIT(1)      // used by UART0
#define P02_UNUSED_BIT     BIT(2)      // P0.02 unused - low output
#define P03_UNUSED_BIT     BIT(3)      // P0.03 unused - low output
#define P04_UNUSED_BIT     BIT(4)      // P0.04 unused - low output
#define P05_UNUSED_BIT     BIT(5)      // P0.05 unused - low output
#define P06_UNUSED_BIT     BIT(6)      // P0.06 unused - low output
#define P07_UNUSED_BIT     BIT(7)      // P0.06 unused - low output
#define P08_UNUSED_BIT     BIT(8)      // P0.08 unused - low output
#define P09_UNUSED_BIT     BIT(9)      // P0.09 unused - low output
#define P10_UNUSED_BIT     BIT(10)     // P0.10 unused - low output
#define P11_UNuSED_BIT     BIT(11)     // P0.11 unused - low output
#define P12_UNUSED_BIT     BIT(12)     // P0.12 unused - low output
#define P13_UNUSED_BIT     BIT(13)     // P0.13 unused - low output
#define P14_UNUSED_BIT     BIT(14)     // P0.14 unused - low output
#define P15_UNUSED_BIT     BIT(15)     // P0.15 unused - low output
#define P16_UNUSED_BIT     BIT(16)     // P0.16 unused - low output
#define P17_UNUSED_BIT     BIT(17)     // P0.17 unused - low output
#define P18_UNUSED_BIT     BIT(18)     // P0.18 unused - low output
#define P19_UNUSED_BIT     BIT(19)     // P0.19 unused - low output
#define P20_UNUSED_BIT     BIT(20)     // P0.20 unused - low output
#define P21_UNUSED_BIT     BIT(21)     // P0.21 unused - low output
#define P22_UNUSED_BIT     BIT(22)     // P0.22 unused - low output
#define P23_UNUSED_BIT     BIT(23)     // P0.23 unused - low output
#define P24_UNUSED_BIT     BIT(24)     // P0.24 unused - low output
#define P25_UNUSED_BIT     BIT(25)     // P0.25 unused - low output
#define P26_UNUSED_BIT     BIT(26)     // P0.26 unused - low output
#define P27_UNUSED_BIT     BIT(27)     // P0.27 unused - low output
#define P28_UNUSED_BIT     BIT(28)     // P0.28 unused - low output
#define P29_UNUSED_BIT     BIT(29)     // P0.29 unused - low output
```

```
#define P30_UNUSED_BIT      BIT(30)     // P0.30 unused - low output
#define P31_UNUSED_BIT      BIT(31)     // P0.31 unused - low output

#define PIO_ZERO_BITS       (uint32_t) ( \
                                P02_UNUSED_BIT | \
                                P03_UNUSED_BIT | \
                                P04_UNUSED_BIT | \
                                P05_UNUSED_BIT | \
                                P06_UNUSED_BIT | \
                                P07_UNUSED_BIT | \
                                P08_UNUSED_BIT | \
                                P09_UNUSED_BIT | \
                                P10_UNUSED_BIT | \
                                P11_UNUSED_BIT | \
                                P12_UNUSED_BIT | \
                                P13_UNUSED_BIT | \
                                P14_UNUSED_BIT | \
                                P15_UNUSED_BIT | \
                                P16_UNUSED_BIT | \
                                P17_UNUSED_BIT | \
                                P18_UNUSED_BIT | \
                                P19_UNUSED_BIT | \
                                P20_UNUSED_BIT | \
                                P21_UNUSED_BIT | \
                                P22_UNUSED_BIT | \
                                P23_UNUSED_BIT | \
                                P24_UNUSED_BIT | \
                                P25_UNUSED_BIT | \
                                P26_UNUSED_BIT | \
                                P27_UNUSED_BIT | \
                                P28_UNUSED_BIT | \
                                P29_UNUSED_BIT | \
                                P30_UNUSED_BIT | \
                                P31_UNUSED_BIT | \
                                0 )

#define PIO_OUTPUT_BITS     (uint32_t) ( \
                               PIO_ZERO_BITS)

#endif
```

armVIC.h

```
/******************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface definitions for setting up and
 * controlling the various interrupt modes present on the ARM processor.
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_ARM_VIC_H
#define INC_ARM_VIC_H

/******************************************************************************
 *
 * MACRO Name: ISR_ENTRY()
 *
 * Description:
 *    This MACRO is used upon entry to an ISR.  The current version of
```

```
 *    the gcc compiler for ARM does not produce correct code for
 *    interrupt routines to operate properly with THUMB code.  The MACRO
 *    performs the following steps:
 *
 *    1 - Adjust address at which execution should resume after servicing
 *        ISR to compensate for IRQ entry
 *    2 - Save the non-banked registers r0-r12 and lr onto the IRQ stack.
 *    3 - Get the status of the interrupted program is in SPSR.
 *    4 - Push it onto the IRQ stack as well.
 *
 ******************************************************************************/
#define ISR_ENTRY() asm volatile(" sub   lr, lr,#4\n" \
                         " stmfd sp!,{r0-r12,lr}\n" \
                         " mrs   r1, spsr\n" \
                         " stmfd sp!,{r1}")

/******************************************************************************
 *
 * MACRO Name: ISR_EXIT()
 *
 * Description:
 *    This MACRO is used to exit an ISR.  The current version of the gcc
 *    compiler for ARM does not produce correct code for interrupt
 *    routines to operate properly with THUMB code.  The MACRO performs
 *    the following steps:
 *
 *    1 - Recover SPSR value from stack
 *    2 - and restore  its value
 *    3 - Pop the return address & the saved general registers from
 *        the IRQ stack & return
 *
 ******************************************************************************/
#define ISR_EXIT()  asm volatile(" ldmfd sp!,{r1}\n" \
                         " msr   spsr_c,r1\n" \
                         " ldmfd sp!,{r0-r12,pc}^")

/******************************************************************************
 *
 * Function Name: disableIRQ()
 *
 * Description:
 *    This function sets the IRQ disable bit in the status register
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    previous value of CPSR
 *
 ******************************************************************************/
unsigned disableIRQ(void);

/******************************************************************************
 *
 * Function Name: enableIRQ()
 *
 * Description:
 *    This function clears the IRQ disable bit in the status register
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    previous value of CPSR
 *
 ******************************************************************************/
```

```
unsigned enableIRQ(void);

/*****************************************************************************
 *
 * Function Name: restoreIRQ()
 *
 * Description:
 *    This function restores the IRQ disable bit in the status register
 *    to the value contained within passed oldCPSR
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    previous value of CPSR
 *
 *****************************************************************************/
unsigned restoreIRQ(unsigned oldCPSR);

/*****************************************************************************
 *
 * Function Name: disableFIQ()
 *
 * Description:
 *    This function sets the FIQ disable bit in the status register
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    previous value of CPSR
 *
 *****************************************************************************/
unsigned disableFIQ(void);

/*****************************************************************************
 *
 * Function Name: enableFIQ()
 *
 * Description:
 *    This function clears the FIQ disable bit in the status register
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    previous value of CPSR
 *
 *****************************************************************************/
unsigned enableFIQ(void);

/*****************************************************************************
 *
 * Function Name: restoreFIQ()
 *
 * Description:
 *    This function restores the FIQ disable bit in the status register
 *    to the value contained within passed oldCPSR
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    previous value of CPSR
 *
 *****************************************************************************/
```

```
unsigned restoreFIQ(unsigned oldCPSR);

#endif
```

armVIC.c

```
/****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface routines for setting up and
 * controlling the various interrupt modes present on the ARM processor.
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 ****************************************************************************/
#include "types.h"
#include "armVIC.h"

#define IRQ_MASK 0x00000080
#define FIQ_MASK 0x00000040
#define INT_MASK (IRQ_MASK | FIQ_MASK)

static inline unsigned __get_cpsr(void)
{
  unsigned long retval;
  asm volatile (" mrs  %0, cpsr" : "=r" (retval) : /* no inputs */  );
  return retval;
}

static inline void __set_cpsr(unsigned val)
{
  asm volatile (" msr  cpsr, %0" : /* no outputs */ : "r" (val)  );
}

unsigned disableIRQ(void)
{
  unsigned _cpsr;

  _cpsr = __get_cpsr();
  __set_cpsr(_cpsr | IRQ_MASK);
  return _cpsr;
}

unsigned restoreIRQ(unsigned oldCPSR)
{
  unsigned _cpsr;

  _cpsr = __get_cpsr();
  __set_cpsr((_cpsr & ~IRQ_MASK) | (oldCPSR & IRQ_MASK));
  return _cpsr;
}

unsigned enableIRQ(void)
{
  unsigned _cpsr;

  _cpsr = __get_cpsr();
  __set_cpsr(_cpsr & ~IRQ_MASK);
  return _cpsr;
}
```

```
unsigned disableFIQ(void)
{
  unsigned _cpsr;

  _cpsr = __get_cpsr();
  __set_cpsr(_cpsr | FIQ_MASK);
  return _cpsr;
}

unsigned restoreFIQ(unsigned oldCPSR)
{
  unsigned _cpsr;

  _cpsr = __get_cpsr();
  __set_cpsr((_cpsr & ~FIQ_MASK) | (oldCPSR & FIQ_MASK));
  return _cpsr;
}

unsigned enableFIQ(void)
{
  unsigned _cpsr;

  _cpsr = __get_cpsr();
  __set_cpsr(_cpsr & ~FIQ_MASK);
  return _cpsr;
}
```

uart.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface definitions for for uart.c
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_UART_H
#define INC_UART_H

#include "types.h"
#include "LPC21xx.h"
#include "config.h"

////////////////////////////////////////////////////////////////////////////
#define UART0_SUPPORT    (1)          // non-zero to enable UART0 code
#define UART1_SUPPORT    (1)          // non-zero to enable UART1 code

////////////////////////////////////////////////////////////////////////////
// uncomment the following to use various interrupt modes
#define UART0_INT_MODE
//#define UART1_INT_MODE
// or
//#define UART0_TX_INT_MODE
//#define UART0_RX_INT_MODE
//#define UART1_TX_INT_MODE
//#define UART1_RX_INT_MODE

////////////////////////////////////////////////////////////////////////////
// code is optimized for power of 2 buffer sizes (16, 32, 64, 128, ...)
// NOTE: the buffers are only used if the respective interrupt mode is
```

```
// enabled
#define UART0_RX_BUFFER_SIZE 64         // UART0 receive buffer size
#define UART0_TX_BUFFER_SIZE 128        // UART0 transmit buffer size
#define UART1_RX_BUFFER_SIZE 128        // UART1 receive buffer size
#define UART1_TX_BUFFER_SIZE 128        // UART1 transmit buffer size

//////////////////////////////////////////////////////////////////////////////
// use the following macros to determine the 'baud' parameter values
// for uart0Init() and uart1Init()
// CAUTION - 'baud' SHOULD ALWAYS BE A CONSTANT or
// a lot of code will be generated.
#define UART_BAUD(baud) (uint16_t)((PCLK / ((baud) * 16.0)) + 0.5)

//////////////////////////////////////////////////////////////////////////////
// Definitions for typical UART 'baud' settings
#define B1200         UART_BAUD(1200)
#define B9600         UART_BAUD(9600)
#define B19200        UART_BAUD(19200)
#define B38400        UART_BAUD(38400)
#define B57600        UART_BAUD(57600)
#define B115200       UART_BAUD(115200)

//////////////////////////////////////////////////////////////////////////////
// Definitions for typical UART 'mode' settings
#define UART_8N1      (uint8_t)(ULCR_CHAR_8 + ULCR_PAR_NO   + ULCR_STOP_1)
#define UART_7N1      (uint8_t)(ULCR_CHAR_7 + ULCR_PAR_NO   + ULCR_STOP_1)
#define UART_8N2      (uint8_t)(ULCR_CHAR_8 + ULCR_PAR_NO   + ULCR_STOP_2)
#define UART_7N2      (uint8_t)(ULCR_CHAR_7 + ULCR_PAR_NO   + ULCR_STOP_2)
#define UART_8E1      (uint8_t)(ULCR_CHAR_8 + ULCR_PAR_EVEN + ULCR_STOP_1)
#define UART_7E1      (uint8_t)(ULCR_CHAR_7 + ULCR_PAR_EVEN + ULCR_STOP_1)
#define UART_8E2      (uint8_t)(ULCR_CHAR_8 + ULCR_PAR_EVEN + ULCR_STOP_2)
#define UART_7E2      (uint8_t)(ULCR_CHAR_7 + ULCR_PAR_EVEN + ULCR_STOP_2)
#define UART_8O1      (uint8_t)(ULCR_CHAR_8 + ULCR_PAR_ODD  + ULCR_STOP_1)
#define UART_7O1      (uint8_t)(ULCR_CHAR_7 + ULCR_PAR_ODD  + ULCR_STOP_1)
#define UART_8O2      (uint8_t)(ULCR_CHAR_8 + ULCR_PAR_ODD  + ULCR_STOP_2)
#define UART_7O2      (uint8_t)(ULCR_CHAR_7 + ULCR_PAR_ODD  + ULCR_STOP_2)

//////////////////////////////////////////////////////////////////////////////
// Definitions for typical UART 'fmode' settings
#define UART_FIFO_OFF (0x00)
#define UART_FIFO_1   (uint8_t)(UFCR_FIFO_ENABLE + UFCR_FIFO_TRIG1)
#define UART_FIFO_4   (uint8_t)(UFCR_FIFO_ENABLE + UFCR_FIFO_TRIG4)
#define UART_FIFO_8   (uint8_t)(UFCR_FIFO_ENABLE + UFCR_FIFO_TRIG8)
#define UART_FIFO_14  (uint8_t)(UFCR_FIFO_ENABLE + UFCR_FIFO_TRIG14)


//////////////////////////////////////////////////////////////////////////////
#if UART0_SUPPORT

#ifdef UART0_INT_MODE
#ifndef UART0_TX_INT_MODE
#define UART0_TX_INT_MODE
#endif // UART0_TX_INT_MODE

#ifndef UART0_RX_INT_MODE
#define UART0_RX_INT_MODE
#endif // UART0_RX_INT_MODE
#endif // UART0_INT_MODE

/*****************************************************************************
 *
 * Function Name: uart0Init()
 *
 * Description:
 *    This function initializes the UART for async mode
 *
```

```
 * Calling Sequence:
 *    baudrate divisor - use UART0_BAUD macro
 *    mode - see typical modes (above)
 *    fmode - see typical fmodes (above)
 *
 * Returns:
 *    void
 *
 * NOTE: uart0Init(UART_BAUD(9600), UART_8N1, UART_FIFO_8);
 *
 ******************************************************************************/
void uart0Init(uint16_t baud, uint8_t mode, uint8_t fmode);

/******************************************************************************
 *
 * Function Name: uart0Putch()
 *
 * Description:
 *    This function puts a character into the UART output queue for
 *    transmission.
 *
 * Calling Sequence:
 *    character to be transmitted
 *
 * Returns:
 *    ch on success, -1 on error (queue full)
 *
 ******************************************************************************/
int uart0Putch(int ch);

/******************************************************************************
 *
 * Function Name: uart0Space()
 *
 * Description:
 *    This function gets the available space in the transmit queue
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    available space in the transmit queue
 *
 ******************************************************************************/
uint16_t uart0Space(void);

/******************************************************************************
 *
 * Function Name: uart0Puts()
 *
 * Description:
 *    This function writes a NULL terminated 'string' to the UART output
 *    queue, returning a pointer to the next character to be written.
 *
 * Calling Sequence:
 *    address of the string
 *
 * Returns:
 *    a pointer to the next character to be written
 *    (\0 if full string is written)
 *
 ******************************************************************************/
const char *uart0Puts(const char *string);

/******************************************************************************
 *
```

```
 * Function Name: uart0Write()
 *
 * Description:
 *    This function writes 'count' characters from 'buffer' to the UART
 *    output queue.
 *
 * Calling Sequence:
 *
 *
 * Returns:
 *    0 on success, -1 if insufficient room, -2 on error
 *    NOTE: if insufficient room, no characters are written.
 *
 ******************************************************************************/
int uart0Write(const char *buffer, uint16_t count);

/******************************************************************************
 *
 * Function Name: uart0TxEmpty()
 *
 * Description:
 *    This function returns the status of the UART transmit data
 *    registers.
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    FALSE - either the tx holding or shift register is not empty
 *   !FALSE - if both the tx holding & shift registers are empty
 *
 ******************************************************************************/
int uart0TxEmpty(void);

/******************************************************************************
 *
 * Function Name: uart0TxFlush()
 *
 * Description:
 *    This function removes all characters from the UART transmit queue
 *    (without transmitting them).
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 ******************************************************************************/
void uart0TxFlush(void);

/******************************************************************************
 *
 * Function Name: uart0Getch()
 *
 * Description:
 *    This function gets a character from the UART receive queue
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    character on success, -1 if no character is available
 *
 ******************************************************************************/
int uart0Getch(void);
```

```
#endif


////////////////////////////////////////////////////////////////////////////
#if UART1_SUPPORT

#ifdef UART1_INT_MODE
#ifndef UART1_TX_INT_MODE
#define UART1_TX_INT_MODE
#endif // UART1_TX_INT_MODE

#ifndef UART1_RX_INT_MODE
#define UART1_RX_INT_MODE
#endif // UART1_RX_INT_MODE
#endif // UART1_INT_MODE

/*****************************************************************************
 *
 * Function Name: uart1Init()
 *
 * Description:
 *    This function initializes the UART for async mode
 *
 * Calling Sequence:
 *    baudrate divisor - use UART_BAUD macro
 *    mode - see typical modes (above)
 *    fmode - see typical fmodes (above)
 *
 * Returns:
 *    void
 *
 * NOTE: uart1Init(UART_BAUD(9600), UART_8N1, UART_FIFO_8);
 *
 *****************************************************************************/
void uart1Init(uint16_t baud, uint8_t mode, uint8_t fmode);

/*****************************************************************************
 *
 * Function Name: uart1Putch()
 *
 * Description:
 *    This function puts a character into the UART output queue for
 *    transmission.
 *
 * Calling Sequence:
 *    character to be transmitted
 *
 * Returns:
 *    ch on success, -1 on error (queue full)
 *
 *****************************************************************************/
int uart1Putch(int ch);

/*****************************************************************************
 *
 * Function Name: uart1Space()
 *
 * Description:
 *    This function gets the available space in the transmit queue
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    available space in the transmit queue
```

```
 *
 ******************************************************************************/
uint16_t uart1Space(void);

/******************************************************************************
 *
 * Function Name: uart1Puts()
 *
 * Description:
 *    This function writes a NULL terminated 'string' to the UART output
 *    queue, returning a pointer to the next character to be written.
 *
 * Calling Sequence:
 *    address of the string
 *
 * Returns:
 *    a pointer to the next character to be written
 *    (\0 if full string is written)
 *
 ******************************************************************************/
const char *uart1Puts(const char *string);

/******************************************************************************
 *
 * Function Name: uart1Write()
 *
 * Description:
 *    This function writes 'count' characters from 'buffer' to the UART
 *    output queue.
 *
 * Calling Sequence:
 *
 *
 * Returns:
 *    0 on success, -1 if insufficient room, -2 on error
 *    NOTE: if insufficient room, no characters are written.
 *
 ******************************************************************************/
int uart1Write(const char *buffer, uint16_t count);

/******************************************************************************
 *
 * Function Name: uart1TxEmpty()
 *
 * Description:
 *    This function returns the status of the UART transmit data
 *    registers.
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    FALSE - either the tx holding or shift register is not empty
 *   !FALSE - if both the tx holding & shift registers are empty
 *
 ******************************************************************************/
int uart1TxEmpty(void);

/******************************************************************************
 *
 * Function Name: uart1TxFlush()
 *
 * Description:
 *    This function removes all characters from the UART transmit queue
 *    (without transmitting them).
 *
```

```
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 ****************************************************************************/
void uart1TxFlush(void);

/****************************************************************************
 *
 * Function Name: uart1Getch()
 *
 * Description:
 *    This function gets a character from the UART receive queue
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    character on success, -1 if no character is available
 *
 ****************************************************************************/
int uart1Getch(void);

#endif

#endif
```

uart.c

```
/****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides interface routines to the LPC ARM UARTs.
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 ****************************************************************************/
#include <limits.h>
#include "types.h"
#include "LPC21xx.h"
#include "uart.h"

#if defined(UART0_TX_INT_MODE) || defined(UART0_RX_INT_MODE) || \
    defined(UART1_TX_INT_MODE) || defined(UART1_RX_INT_MODE)
#include "armVIC.h"
#include "uartISR.h"
#endif

#if UART0_SUPPORT
#ifdef UART0_RX_INT_MODE
uint8_t  uart0_rx_buffer[UART0_RX_BUFFER_SIZE];
uint16_t uart0_rx_insert_idx, uart0_rx_extract_idx;
#endif

#ifdef UART0_TX_INT_MODE
uint8_t  uart0_tx_buffer[UART0_TX_BUFFER_SIZE];
uint16_t uart0_tx_insert_idx, uart0_tx_extract_idx;
int      uart0_tx_running;
#endif
```

```
#endif

#if UART1_SUPPORT
#ifdef UART1_RX_INT_MODE
uint8_t  uart1_rx_buffer[UART1_RX_BUFFER_SIZE];
uint16_t uart1_rx_insert_idx, uart1_rx_extract_idx;
#endif

#ifdef UART1_TX_INT_MODE
uint8_t  uart1_tx_buffer[UART1_TX_BUFFER_SIZE];
uint16_t uart1_tx_insert_idx, uart1_tx_extract_idx;
int      uart1_tx_running;
#endif
#endif

#if UART0_SUPPORT


void uart0Init(uint16_t baud, uint8_t mode, uint8_t fmode)
{
  // set port pins for UART0
  PINSEL0 = (PINSEL0 & ~U0_PINMASK) | U0_PINSEL;

  U0IER = 0x00;                       // disable all interrupts
  U0IIR;                             // clear interrupt ID
  U0RBR;                               // clear receive register
  U0LSR;                               // clear line status register

  // set the baudrate
  U0LCR = ULCR_DLAB_ENABLE;            // select divisor latches
  U0DLL = (uint8_t)baud;               // set for baud low byte
  U0DLM = (uint8_t)(baud >> 8);        // set for baud high byte

  // set the number of characters and other
  // user specified operating parameters
  U0LCR = (mode & ~ULCR_DLAB_ENABLE);
  U0FCR = fmode;

#if defined(UART0_TX_INT_MODE) || defined(UART0_RX_INT_MODE)
  // initialize the interrupt vector
  VICIntSelect &= ~VIC_BIT(VIC_UART0);  // UART0 selected as IRQ
  VICIntEnable = VIC_BIT(VIC_UART0);    // UART0 interrupt enabled
  VICVectCntl0 = VIC_ENABLE | VIC_UART0;
  VICVectAddr0 = (uint32_t)uart0ISR;    // address of the ISR

#ifdef UART0_TX_INT_MODE
  // initialize the transmit data queue
  uart0_tx_extract_idx = uart0_tx_insert_idx = 0;
  uart0_tx_running = 0;
#endif

#ifdef UART0_RX_INT_MODE
  // initialize the receive data queue
  uart0_rx_extract_idx = uart0_rx_insert_idx = 0;

  // enable receiver interrupts
  U0IER = UIER_ERBFI;
#endif
#endif
}


int uart0Putch(int ch)
{
#ifdef UART0_TX_INT_MODE
  uint16_t temp;
```

```
  unsigned cpsr;

  temp = (uart0_tx_insert_idx + 1) % UART0_TX_BUFFER_SIZE;

  if (temp == uart0_tx_extract_idx)
    return -1;                      // no room

  cpsr = disableIRQ();              // disable global interrupts
  U0IER &= ~UIER_ETBEI;             // disable TX interrupts
  restoreIRQ(cpsr);                 // restore global interrupts

  // check if in process of sending data
  if (uart0_tx_running)
    {
    // add to queue
    uart0_tx_buffer[uart0_tx_insert_idx] = (uint8_t)ch;
    uart0_tx_insert_idx = temp;
    }
  else
    {
    // set running flag and write to output register
    uart0_tx_running = 1;
    U0THR = (uint8_t)ch;
    }

  cpsr = disableIRQ();              // disable global interrupts
  U0IER |= UIER_ETBEI;              // enable TX interrupts
  restoreIRQ(cpsr);                 // restore global interrupts
#else
  while (!(U0LSR & ULSR_THRE))      // wait for TX buffer to empty
    continue;                       // also either WDOG() or swap()

  U0THR = (uint8_t)ch;
#endif
  return (uint8_t)ch;
}


uint16_t uart0Space(void)
{
#ifdef UART0_TX_INT_MODE
  int space;

  if ((space = (uart0_tx_extract_idx - uart0_tx_insert_idx)) <= 0)
    space += UART0_TX_BUFFER_SIZE;

  return (uint16_t)(space - 1);
#else
  return USHRT_MAX;
#endif
}


const char *uart0Puts(const char *string)
{
  register char ch;

  while ((ch = *string) && (uart0Putch(ch) >= 0))
    string++;

  return string;
}


int uart0Write(const char *buffer, uint16_t count)
{
```

```
#ifdef UART0_TX_INT_MODE
  if (count > uart0Space())
    return -1;
#endif
  while (count && (uart0Putch(*buffer++) >= 0))
    count--;

  return (count ? -2 : 0);
}


int uart0TxEmpty(void)
{
  return (U0LSR & (ULSR_THRE | ULSR_TEMT)) == (ULSR_THRE | ULSR_TEMT);
}


void uart0TxFlush(void)
{
#ifdef UART0_TX_INT_MODE
  unsigned cpsr;

  U0FCR |= UFCR_TX_FIFO_RESET;          // clear the TX fifo

  // "Empty" the transmit buffer.
  cpsr = disableIRQ();                  // disable global interrupts
  U0IER &= ~UIER_ETBEI;                 // disable TX interrupts
  restoreIRQ(cpsr);                     // restore global interrupts
  uart0_tx_insert_idx = uart0_tx_extract_idx = 0;
#else
  U0FCR |= UFCR_TX_FIFO_RESET;          // clear the TX fifo
#endif
}


int uart0Getch(void)
{
#ifdef UART0_RX_INT_MODE
  uint8_t ch;

  if (uart0_rx_insert_idx == uart0_rx_extract_idx) // check if character is available
    return -1;

  ch = uart0_rx_buffer[uart0_rx_extract_idx++]; // get character, bump pointer
  uart0_rx_extract_idx %= UART0_RX_BUFFER_SIZE; // limit the pointer
  return ch;
#else
  if (U0LSR & ULSR_RDR)                 // check if character is available
    return U0RBR;                       // return character

  return -1;
#endif
}

#endif


#if UART1_SUPPORT


void uart1Init(uint16_t baud, uint8_t mode, uint8_t fmode)
{
  // set port pins for UART1
  PINSEL0 = (PINSEL0 & ~U1_PINMASK) | U1_PINSEL;

  U1IER = 0x00;                         // disable all interrupts
```

```
  U1IIR;                        // clear interrupt ID
  U1RBR;                          // clear receive register
  U1LSR;                          // clear line status register

  // set the baudrate
  U1LCR = ULCR_DLAB_ENABLE;           // select divisor latches
  U1DLL = (uint8_t)baud;              // set for baud low byte
  U1DLM = (uint8_t)(baud >> 8);       // set for baud high byte

  // set the number of characters and other
  // user specified operating parameters
  U1LCR = (mode & ~ULCR_DLAB_ENABLE);
  U1FCR = fmode;

#if defined(UART1_TX_INT_MODE) || defined(UART1_RX_INT_MODE)
  // initialize the interrupt vector
  VICIntSelect &= ~VIC_BIT(VIC_UART1);  // UART1 selected as IRQ
  VICIntEnable = VIC_BIT(VIC_UART1);    // UART1 interrupt enabled
  VICVectCntl1 = VIC_ENABLE | VIC_UART1;
  VICVectAddr1 = (uint32_t)uart1ISR;    // address of the ISR

#ifdef UART1_TX_INT_MODE
  uart1_tx_extract_idx = uart1_tx_insert_idx = 0;
  uart1_tx_running = 0;
#endif

#ifdef UART1_RX_INT_MODE
  // initialize data queues
  uart1_rx_extract_idx = uart1_rx_insert_idx = 0;

  // enable receiver interrupts
  U1IER |= UIER_ERBFI;
#endif
#endif
}


int uart1Putch(int ch)
{
#ifdef UART1_TX_INT_MODE
  uint16_t temp;
  unsigned cpsr;

  temp = (uart1_tx_insert_idx + 1) % UART1_TX_BUFFER_SIZE;

  if (temp == uart1_tx_extract_idx)
    return -1;                      // no room

  cpsr = disableIRQ();                // disable global interrupts
  U1IER &= ~UIER_ETBEI;                 // disable TX interrupts
  restoreIRQ(cpsr);                   // restore global interrupts

  // check if in process of sending data
  if (uart1_tx_running)
    {
    // add to queue
    uart1_tx_buffer[uart1_tx_insert_idx] = (uint8_t)ch;
    uart1_tx_insert_idx = temp;
    }
  else
    {
    // set running flag and write to output register
    uart1_tx_running = 1;
    U1THR = (uint8_t)ch;
    }
```

```
  cpsr = disableIRQ();                  // disable global interrupts
  U1IER |= UIER_ETBEI;                  // enable TX interrupts
  restoreIRQ(cpsr);                     // restore global interrupts
#else
  while (!(U1LSR & ULSR_THRE))          // wait for TX buffer to empty
    continue;                           // also either WDOG() or swap()

  U1THR = (uint8_t)ch;
#endif
  return (uint8_t)ch;
}


uint16_t uart1Space(void)
{
#ifdef UART1_TX_INT_MODE
  int space;

  if ((space = (uart1_tx_extract_idx - uart1_tx_insert_idx)) <= 0)
    space += UART1_TX_BUFFER_SIZE;

  return (uint16_t)(space - 1);
#else
  return USHRT_MAX;
#endif
}


const char *uart1Puts(const char *string)
{
  register char ch;

  while ((ch = *string) && (uart1Putch(ch) >= 0))
    string++;

  return string;
}


int uart1Write(const char *buffer, uint16_t count)
{
#ifdef UART1_TX_INT_MODE
  if (count > uart1Space())
    return -1;
#endif
  while (count && (uart1Putch(*buffer++) >= 0))
    count--;

  return (count ? -2 : 0);
}


int uart1TxEmpty(void)
{
  return (U1LSR & (ULSR_THRE | ULSR_TEMT)) == (ULSR_THRE | ULSR_TEMT);
}


void uart1TxFlush(void)
{
#ifdef UART1_TX_INT_MODE
  unsigned cpsr;

  U1FCR |= UFCR_TX_FIFO_RESET;          // clear the TX fifo

  // "Empty" the transmit buffer.
```

```
  cpsr = disableIRQ();                  // disable global interrupts
  U1IER &= ~UIER_ETBEI;                 // disable TX interrupts
  restoreIRQ(cpsr);                     // restore global interrupts
  uart1_tx_insert_idx = uart1_tx_extract_idx = 0;
#else
  U1FCR |= UFCR_TX_FIFO_RESET;          // clear the TX fifo
#endif
}


int uart1Getch(void)
{
#ifdef UART1_RX_INT_MODE
  uint8_t ch;

  if (uart1_rx_insert_idx == uart1_rx_extract_idx) // check if character is available
    return -1;

  ch = uart1_rx_buffer[uart1_rx_extract_idx++]; // get character, bump pointer
  uart1_rx_extract_idx %= UART1_RX_BUFFER_SIZE; // limit the pointer
  return ch;
#else
  if (U1LSR & ULSR_RDR)                 // check if character is available
    return U1RBR;                       // return character

  return -1;
#endif
}

#endif
```

sysTime.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface definitions for sysTime.c
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#ifndef INC_SYS_TIME_H
#define INC_SYS_TIME_H

#include "types.h"
#include "LPC21xx.h"
#include "config.h"

// Note: with a PCLK = CCLK/2 = 60MHz/2 and a Prescale divider of 3, we
// have a resolution of 100nSec.  Given the timer's counter register is
// 32-bits, we must make a call to one of the sysTime functions at least
// every ~430 sec.

// setup parameters
#define T0_PCLK_DIV     3
#define sysTICSperSEC   (PCLK / T0_PCLK_DIV)

// some helpful times for pause()
#define ONE_MS          (uint32_t)((  1e-3 * sysTICSperSEC) + .5)
#define TWO_MS          (uint32_t)((  2e-3 * sysTICSperSEC) + .5)
#define FIVE_MS         (uint32_t)((  5e-3 * sysTICSperSEC) + .5)
```

```
#define TEN_MS          (uint32_t)(( 10e-3 * sysTICSperSEC) + .5)
#define TWENTY_MS       (uint32_t)(( 20e-3 * sysTICSperSEC) + .5)
#define THIRTY_MS       (uint32_t)(( 30e-3 * sysTICSperSEC) + .5)
#define FIFTY_MS        (uint32_t)(( 50e-3 * sysTICSperSEC) + .5)
#define HUNDRED_MS      (uint32_t)((100e-3 * sysTICSperSEC) + .5)
#define ONE_FIFTY_MS    (uint32_t)((150e-3 * sysTICSperSEC) + .5)
#define QUARTER_SEC     (uint32_t)((250e-3 * sysTICSperSEC) + .5)
#define HALF_SEC        (uint32_t)((500e-3 * sysTICSperSEC) + .5)
#define ONE_SEC         (uint32_t)(( 1.0   * sysTICSperSEC) + .5)
#define TWO_SEC         (uint32_t)(( 2.0   * sysTICSperSEC) + .5)
#define FIVE_SEC        (uint32_t)(( 5.0   * sysTICSperSEC) + .5)
#define TEN_SEC         (uint32_t)((10.0   * sysTICSperSEC) + .5)


/*****************************************************************************
 *
 * Function Name: initSysTime()
 *
 * Description:
 *    This function initializes the LPC's Timer 0 for use as the system
 *    timer.
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 *****************************************************************************/
void initSysTime(void);

/*****************************************************************************
 *
 * Function Name: getSysTICs()
 *
 * Description:
 *    This function returns the current system time in TICs.
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    The current time in TICs
 *
 *****************************************************************************/
uint32_t getSysTICs(void);

/*****************************************************************************
 *
 * Function Name: getElapsedSysTICs()
 *
 * Description:
 *    This function then returns the difference in TICs between the
 *    given starting time and the current system time.
 *
 * Calling Sequence:
 *    The starting time.
 *
 * Returns:
 *    The time difference.
 *
 *****************************************************************************/
uint32_t getElapsedSysTICs(uint32_t startTime);

/*****************************************************************************
 *
```

```
 * Function Name: pause()
 *
 * Description:
 *    This function does not return until the specified 'duration' in
 *    TICs has elapsed.
 *
 * Calling Sequence:
 *    duration - length of time in TICs to wait before returning
 *
 * Returns:
 *    void
 *
 *****************************************************************************/
void pause(uint32_t duration);

#endif
```

sysTime.c

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface routines for initializing and
 * accessing the system timing functions.
 * Copyright 2004, R O SoftWare
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#include "types.h"
#include "LPC21xx.h"
#include "config.h"
#include "sysTime.h"

static uint32_t sysTICs;
static uint32_t lastT0TC;


void initSysTime(void)
{
  // setup Timer 1 to count forever
  T0TCR = TCR_RESET;                  // reset & disable timer 0
  T0PR = T0_PCLK_DIV - 1;             // set the prescale divider
  T0MCR = 0;                          // disable match registers
  T0CCR = 0;                          // disable compare registers
  T0EMR = 0;                          // disable external match register
  T0TCR = TCR_ENABLE;                   // enable timer 0
  sysTICs = 0;
}


uint32_t getSysTICs(void)
{
  uint32_t now = T0TC;

  sysTICs += (uint32_t)(now - lastT0TC);
  lastT0TC = now;
  return sysTICs;
}


uint32_t getElapsedSysTICs(uint32_t startTime)
```

```
{
  return getSysTICs() - startTime;
}


void pause(uint32_t duration)
{
  uint32_t startTime = getSysTICs();

  while (getElapsedSysTICs(startTime) < duration)
    WDOG();
}
```

can.h

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface definitions for can.c
 * Copyright 2007, Alexandros Papanastasatos
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/

#ifndef INC_CAN_H
#define INC_CAN_H

#include "types.h"
#include "LPC21xx.h"
#include "config.h"

////////////////////////////////////////////////////////////////////////////
// Common CAN bit rates
#define   CANBitrate125k_12MHz        0x001C001D
#define   CANBitrate250k_12MHz        0x001C000E

// Type definition to hold a CAN message
typedef struct
{
  uint32_t Frame;             // Bits 0..9: ID Index
                                                  // Bits 16..19: DLC - Data Length Counter
                                                  // Bit 30: Set if this is a RTR message
                                                  // Bit 31: Set if this is a 29-bit ID message
  uint32_t MsgID;             // CAN Message ID (11-bit or 29-bit)
  uint32_t DatA;              // CAN Message Data Bytes 0-3
  uint32_t DatB;              // CAN Message Data Bytes 4-7
} CAN_MSG;

////////////////////////////////////////////////////////////////////////////
/*****************************************************************************
 *
 * Function Name: canInit()
 *
 * Description:
 *    This function initializes the CAN Interface (no interrupts used)
 *
 * Calling Sequence:
 *    can_port - CAN Interface to init (1, 2, 3 or 4)
 *    can_btr - CAN baud rate
 *
 * Returns:
```

```
 *    1 if initialization successful, else zero
 *
 ******************************************************************************/
int canInit(uint8_t can_port, uint32_t can_btr);

/******************************************************************************
 *
 * Function Name: canSend()
 *
 * Description:
 *    This function sends a message through a CAN Interface
 *
 * Calling Sequence:
 *    can_port - CAN Interface to use (1, 2, 3 or 4)
 *    msg - Pointer to the CAN message to send
 *
 * Returns:
 *    1 if message successfully sent, else zero
 *
 ******************************************************************************/
int canSend(uint8_t can_port, CAN_MSG *msg);

/******************************************************************************
 *
 * Function Name: canReceive()
 *
 * Description:
 *    This function receives a CAN message through a CAN Interface
 *
 * Calling Sequence:
 *    can_port - CAN Interface to use (1, 2, 3 or 4)
 *    msg - Pointer to the CAN message received
 *
 * Returns:
 *    1 if message successfully received, else zero
 *
 ******************************************************************************/
int canReceive(uint8_t can_port, CAN_MSG *msg);

#endif
```

Makefile

```
# Hey Emacs, this is a -*- makefile -*-
#
# WinARM template makefile
# by Martin Thomas, Kaiserslautern, Germany
# <eversmith@heizung-thomas.de>
#
# based on the WinAVR makefile written by Eric B. Weddington, Jφrg Wunsch, et al.
# Released to the Public Domain
# Please read the make user manual!
#
#
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make program = Download the hex file to the device, using lpc21isp
#
# (TODO: make filename.s = Just compile filename.c into the assembler code only)
#
# To rebuild project do "make clean" then "make all".
```

```
#
# Changelog:
# - 17. Feb. 2005  - added thumb-interwork support (mth)
# - 28. Apr. 2005  - added C++ support (mth)
# - 29. Arp. 2005  - changed handling for lst-Filename (mth)
#

# MCU name and submodel
MCU = arm7tdmi-s
SUBMDL = LPC2129
THUMB    = -mthumb
THUMB_IW = -mthumb-interwork


## Create ROM-Image (final)
RUN_MODE=ROM_RUN
## Create RAM-Image (debugging)
#RUN_MODE=RAM_RUN


# Output format. (can be srec, ihex, binary)
FORMAT = ihex


# Target file name (without extension).
TARGET = main


# List C source files here. (C dependencies are automatically generated.)
# use file-extension c for "c-only"-files
SRC = $(TARGET).c sysTime.c uart.c can.c

# List C source files here which must be compiled in ARM-Mode.
# use file-extension c for "c-only"-files
SRCARM = uartISR.c armVIC.c

# List C++ source files here.
# use file-extension cpp for C++-files (use extension .cpp)
CPPSRC =

# List C++ source files here which must be compiled in ARM-Mode.
# use file-extension cpp for C++-files (use extension .cpp)
#CPPSRCARM = $(TARGET).cpp
CPPSRCARM =

# List Assembler source files here.
# Make them always end in a capital .S.  Files ending in a lowercase .s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"!
# Even though the DOS/Win* filesystem matches both .s and .S the same,
# it will preserve the spelling of the filenames, and gcc itself does
# care about how the name is spelled on its command-line.
ASRC =

# List Assembler source files here which must be assembled in ARM-Mode..
ASRCARM = crt0.S

# Optimization level, can be [0, 1, 2, 3, s].
# 0 = turn off optimization. s = optimize for size.
# (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = s
#OPT = 0

# Debugging format.
# Native formats for AVR-GCC's -g are stabs [default], or dwarf-2.
# AVR (extended) COFF requires stabs, plus an avr-objcopy run.
```

```
#DEBUG = stabs
DEBUG = dwarf-2

# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
#EXTRAINCDIRS = ./include
EXTRAINCDIRS =

# Compiler flag to set the C Standard level.
# c89   - "ANSI" C
# gnu89 - c89 plus GCC extensions
# c99   - ISO C99 standard (not yet fully implemented)
# gnu99 - c99 plus GCC extensions
CSTANDARD = -std=gnu99

# Place -D or -U options for C here
CDEFS =  -D$(RUN_MODE)

# Place -I options here
CINCS =

# Place -D or -U options for ASM here
ADEFS =  -D$(RUN_MODE)


# Compiler flags.
#  -g*:          generate debugging information
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
#
# Flags for C and C++ (arm-elf-gcc/arm-elf-g++)
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS) $(CINCS)
CFLAGS += -O$(OPT)
CFLAGS += -Wall -Wcast-align -Wcast-qual -Wimplicit
CFLAGS += -Wpointer-arith -Wswitch
CFLAGS += -Wredundant-decls -Wreturn-type -Wshadow -Wunused
CFLAGS += -Wa,-adhlns=$(subst $(suffix $<),.lst,$<)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))

# flags only for C
CONLYFLAGS  = -Wstrict-prototypes -Wmissing-declarations
CONLYFLAGS += -Wmissing-prototypes -Wnested-externs
CONLYFLAGS += $(CSTANDARD)

# flags only for C++ (arm-elf-g++)
# CPPFLAGS = -fno-rtti -fno-exceptions
CPPFLAGS =

# Assembler flags.
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -ahlms:    create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#             files -- see avr-libc docs [FIXME: not yet described there]
##ASFLAGS = -Wa,-adhlns=$(<:.S=.lst),-gstabs
ASFLAGS = $(ADEFS) -Wa,-adhlns=$(<:.S=.lst),-g$(DEBUG)

#Additional libraries.

#Support for newlibc-lpc (file: libnewlibc-lpc.a)
NEWLIBLPC = -lnewlib-lpc
```

```
MATH_LIB = -lm

CPLUSPLUS_LIB = -lstdc++

# Linker flags.
#  -Wl,...:     tell GCC to pass this to linker.
#    -Map:      create map file
#    --cref:    add cross reference to  map file
LDFLAGS = -nostartfiles -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += -lc
LDFLAGS += $(NEWLIBLPC) $(MATH_LIB)
LDFLAGS += -lc -lgcc
LDFLAGS += $(CPLUSPLUS_LIB)

# Set Linker-Script Depending On Selected Memory
ifeq ($(RUN_MODE),RAM_RUN)
LDFLAGS +=-T$(SUBMDL)-RAM.ld
else
LDFLAGS +=-T$(SUBMDL)-ROM.ld
endif



# ----------------------------------------------------------------------------
# Flash-Programming support using lpc21isp by Martin Maurer

# Settings and variables:
#LPC21ISP = lpc21isp
LPC21ISP = lpc21isp_beta
LPC21ISP_PORT = com1
LPC21ISP_BAUD = 115200
LPC21ISP_XTAL = 14746
LPC21ISP_FLASHFILE = $(TARGET).hex
# verbose output:
## LPC21ISP_DEBUG = -debug
# enter bootloader via RS232 DTR/RTS (only if hardware supports this
# feature - see Philips AppNote):
LPC21ISP_CONTROL = -control


# ----------------------------------------------------------------------------

# Define directories, if needed.
## DIRARM = c:/WinARM/
## DIRARMBIN = $(DIRAVR)/bin/
## DIRAVRUTILS = $(DIRAVR)/utils/bin/

# Define programs and commands.
SHELL = sh
CC = arm-elf-gcc
CPP = arm-elf-g++
OBJCOPY = arm-elf-objcopy
OBJDUMP = arm-elf-objdump
SIZE = arm-elf-size
NM = arm-elf-nm
REMOVE = rm -f
COPY = cp


# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
```

```
MSG_SIZE_AFTER = Size after:
MSG_FLASH = Creating load file for Flash:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling C:
MSG_COMPILING_ARM = "Compiling C (ARM-only):"
MSG_COMPILINGCPP = Compiling C++:
MSG_COMPILINGCPP_ARM = "Compiling C++ (ARM-only):"
MSG_ASSEMBLING = Assembling:
MSG_ASSEMBLING_ARM = "Assembling (ARM-only):"
MSG_CLEANING = Cleaning project:
MSG_LPC21_RESETREMINDER = You may have to bring the target in bootloader-mode now.


# Define all object files.
COBJ      = $(SRC:.c=.o)
AOBJ      = $(ASRC:.S=.o)
COBJARM   = $(SRCARM:.c=.o)
AOBJARM   = $(ASRCARM:.S=.o)
CPPOBJ    = $(CPPSRC:.cpp=.o)
CPPOBJARM = $(CPPSRCARM:.cpp=.o)

# Define all listing files.
LST = $(ASRC:.S=.lst) $(ASRCARM:.S=.lst) $(SRC:.c=.lst) $(SRCARM:.c=.lst)
LST += $(CPPSRC:.cpp=.lst) $(CPPSRCARM:.cpp=.lst)

# Compiler flags to generate dependency files.
### GENDEPFLAGS = -Wp,-M,-MP,-MT,$(*F).o,-MF,.dep/$(@F).d
GENDEPFLAGS = -MD -MP -MF .dep/$(@F).d

# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mcpu=$(MCU) $(THUMB_IW) -I. $(CFLAGS) $(GENDEPFLAGS)
ALL_ASFLAGS = -mcpu=$(MCU) $(THUMB_IW) -I. -x assembler-with-cpp $(ASFLAGS)


# Default target.
all: begin gccversion sizebefore build sizeafter finished end

build: elf hex lss sym

elf: $(TARGET).elf
hex: $(TARGET).hex
lss: $(TARGET).lss
sym: $(TARGET).sym

# Eye candy.
begin:
	@echo
	@echo $(MSG_BEGIN)

finished:
	@echo $(MSG_ERRORS_NONE)

end:
	@echo $(MSG_END)
	@echo


# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) -A $(TARGET).elf
sizebefore:
	@if [ -f $(TARGET).elf ]; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE); echo; fi
```

```
sizeafter:
	@if [ -f $(TARGET).elf ]; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE); echo; fi


# Display compiler version information.
gccversion :
	@$(CC) --version


# Program the device.
program: $(TARGET).hex
	@echo
	@echo $(MSG_LPC21_RESETREMINDER)
	$(LPC21ISP) $(LPC21ISP_CONTROL) $(LPC21ISP_DEBUG) $(LPC21ISP_FLASHFILE)
$(LPC21ISP_PORT) $(LPC21ISP_BAUD) $(LPC21ISP_XTAL)


# Create final output files (.hex, .eep) from ELF output file.
# TODO: handling the .eeprom-section should be redundant
%.hex: %.elf
	@echo
	@echo $(MSG_FLASH) $@
	$(OBJCOPY) -O $(FORMAT) $< $@


# Create extended listing file from ELF output file.
# testing: option -C
%.lss: %.elf
	@echo
	@echo $(MSG_EXTENDED_LISTING) $@
	$(OBJDUMP) -h -S -C $< > $@


# Create a symbol table from ELF output file.
%.sym: %.elf
	@echo
	@echo $(MSG_SYMBOL_TABLE) $@
	$(NM) -n $< > $@


# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(AOBJARM) $(AOBJ) $(COBJARM) $(COBJ) $(CPPOBJ) $(CPPOBJARM)
%.elf:  $(AOBJARM) $(AOBJ) $(COBJARM) $(COBJ) $(CPPOBJ) $(CPPOBJARM)
	@echo
	@echo $(MSG_LINKING) $@
	$(CC) $(THUMB) $(ALL_CFLAGS) $(AOBJARM) $(AOBJ) $(COBJARM) $(COBJ) $(CPPOBJ)
$(CPPOBJARM) --output $@ $(LDFLAGS)
#	$(CPP) $(THUMB) $(ALL_CFLAGS) $(AOBJARM) $(AOBJ) $(COBJARM) $(COBJ) $(CPPOBJ)
$(CPPOBJARM) --output $@ $(LDFLAGS)

# Compile: create object files from C source files. ARM/Thumb
$(COBJ) : %.o : %.c
	@echo
	@echo $(MSG_COMPILING) $<
	$(CC) -c $(THUMB) $(ALL_CFLAGS) $(CONLYFLAGS) $< -o $@

# Compile: create object files from C source files. ARM-only
$(COBJARM) : %.o : %.c
	@echo
	@echo $(MSG_COMPILING_ARM) $<
	$(CC) -c $(ALL_CFLAGS) $(CONLYFLAGS) $< -o $@

# Compile: create object files from C++ source files. ARM/Thumb
$(CPPOBJ) : %.o : %.cpp
	@echo
```

```
	@echo $(MSG_COMPILINGCPP) $<
	$(CPP) -c $(THUMB) $(ALL_CFLAGS) $(CPPFLAGS) $< -o $@

# Compile: create object files from C++ source files. ARM-only
$(CPPOBJARM) : %.o : %.cpp
	@echo
	@echo $(MSG_COMPILINGCPP_ARM) $<
	$(CPP) -c $(ALL_CFLAGS) $(CPPFLAGS) $< -o $@


# Compile: create assembler files from C source files. ARM/Thumb
## does not work - TODO - hints welcome
##$(COBJ) : %.s : %.c
##	$(CC) $(THUMB) -S $(ALL_CFLAGS) $< -o $@


# Assemble: create object files from assembler source files. ARM/Thumb
$(AOBJ) : %.o : %.S
	@echo
	@echo $(MSG_ASSEMBLING) $<
	$(CC) -c $(THUMB) $(ALL_ASFLAGS) $< -o $@


# Assemble: create object files from assembler source files. ARM-only
$(AOBJARM) : %.o : %.S
	@echo
	@echo $(MSG_ASSEMBLING_ARM) $<
	$(CC) -c $(ALL_ASFLAGS) $< -o $@


# Target: clean project.
clean: begin clean_list finished end


clean_list :
	@echo
	@echo $(MSG_CLEANING)
	$(REMOVE) $(TARGET).hex
	$(REMOVE) $(TARGET).obj
	$(REMOVE) $(TARGET).elf
	$(REMOVE) $(TARGET).map
	$(REMOVE) $(TARGET).obj
	$(REMOVE) $(TARGET).a90
	$(REMOVE) $(TARGET).sym
	$(REMOVE) $(TARGET).lnk
	$(REMOVE) $(TARGET).lss
	$(REMOVE) $(COBJ)
	$(REMOVE) $(CPPOBJ)
	$(REMOVE) $(AOBJ)
	$(REMOVE) $(COBJARM)
	$(REMOVE) $(CPPOBJARM)
	$(REMOVE) $(AOBJARM)
	$(REMOVE) $(LST)
	$(REMOVE) $(SRC:.c=.s)
	$(REMOVE) $(SRC:.c=.d)
	$(REMOVE) $(SRCARM:.c=.s)
	$(REMOVE) $(SRCARM:.c=.d)
	$(REMOVE) $(CPPSRC:.cpp=.s)
	$(REMOVE) $(CPPSRC:.cpp=.d)
	$(REMOVE) $(CPPSRCARM:.cpp=.s)
	$(REMOVE) $(CPPSRCARM:.cpp=.d)
	$(REMOVE) .dep/*


# Include the dependency files.
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)
```

```
# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex lss sym clean clean_list program
```

crt0.S

```
/*
 crt0.S for LPC2xxx
 - based on examples from R O Software
 - based on examples from newlib-lpc
 - based on an example from Anglia Designs

 collected and modified by Martin Thomas
*/

        .global _etext              // -> .data initial values in ROM
        .global _data               // -> .data area in RAM
        .global _edata               // end of .data area
        .global __bss_start           // -> .bss area in RAM
        .global __bss_end__            // end of .bss area
        .global _stack              // top of stack

// Stack Sizes
        .set  UND_STACK_SIZE, 0x00000004
        .set  ABT_STACK_SIZE, 0x00000004
        .set  FIQ_STACK_SIZE, 0x00000004
        .set  IRQ_STACK_SIZE, 0X00000080
        .set  SVC_STACK_SIZE, 0x00000004

// Standard definitions of Mode bits and Interrupt (I & F) flags in PSRs
        .set  MODE_USR, 0x10           // User Mode
        .set  MODE_FIQ, 0x11           // FIQ Mode
        .set  MODE_IRQ, 0x12           // IRQ Mode
        .set  MODE_SVC, 0x13           // Supervisor Mode
        .set  MODE_ABT, 0x17           // Abort Mode
        .set  MODE_UND, 0x1B           // Undefined Mode
        .set  MODE_SYS, 0x1F           // System Mode

        .equ  I_BIT, 0x80             // when I bit is set, IRQ is disabled
        .equ  F_BIT, 0x40              // when F bit is set, FIQ is disabled

        .text
             .arm
             .section .init, "ax"

        .code 32
        .align 2

        .global _boot
        .func   _boot
_boot:

// Runtime Interrupt Vectors
// ------------------------
Vectors:
        b     _start                 // reset - _start
        ldr   pc,_undf                 // undefined - _undf
        ldr   pc,_swi                  // SWI - _swi
        ldr   pc,_pabt                 // program abort - _pabt
        ldr   pc,_dabt                 // data abort - _dabt
        nop                           // reserved
        ldr   pc,[pc,#-0xFF0]          // IRQ - read the VIC
        ldr   pc,_fiq                  // FIQ - _fiq
```

```
#if 0
// Use this group for production
_undf:  .word _reset                // undefined - _reset
_swi:   .word _reset                // SWI - _reset
_pabt:  .word _reset                // program abort - _reset
_dabt:  .word _reset                // data abort - _reset
_irq:   .word _reset                // IRQ - _reset
_fiq:   .word _reset                // FIQ - _reset

#else
// Use this group for development
_undf:  .word __undf                // undefined
_swi:   .word __swi                 // SWI
_pabt:  .word __pabt                // program abort
_dabt:  .word __dabt                // data abort
_irq:   .word __irq                 // IRQ
_fiq:   .word __fiq                 // FIQ

__undf: b     .                     // undefined
__swi:  b     .                     // SWI
__pabt: b     .                     // program abort
__dabt: b     .                     // data abort
__irq:  b     .                     // IRQ
__fiq:  b     .                     // FIQ
#endif
        .size _boot, . - _boot
        .endfunc


// Setup the operating mode & stack.
// ----------------------------------
        .global _start, start, _mainCRTStartup
        .func   _start

_start:
start:
_mainCRTStartup:

// Initialize Interrupt System
// - Set stack location for each mode
// - Leave in System Mode with Interrupts Disabled
// ------------------------------------------------
        ldr   r0,=_stack
        msr   CPSR_c,#MODE_UND|I_BIT|F_BIT // Undefined Instruction Mode
        mov   sp,r0
        sub   r0,r0,#UND_STACK_SIZE
        msr   CPSR_c,#MODE_ABT|I_BIT|F_BIT // Abort Mode
        mov   sp,r0
        sub   r0,r0,#ABT_STACK_SIZE
        msr   CPSR_c,#MODE_FIQ|I_BIT|F_BIT // FIQ Mode
        mov   sp,r0
        sub   r0,r0,#FIQ_STACK_SIZE
        msr   CPSR_c,#MODE_IRQ|I_BIT|F_BIT // IRQ Mode
        mov   sp,r0
        sub   r0,r0,#IRQ_STACK_SIZE
        msr   CPSR_c,#MODE_SVC|I_BIT|F_BIT // Supervisor Mode
        mov   sp,r0
        sub   r0,r0,#SVC_STACK_SIZE
        msr   CPSR_c,#MODE_SYS|I_BIT|F_BIT // System Mode
        mov   sp,r0

// Copy initialized data to its execution address in RAM
// ------------------------------------------------------
#ifdef ROM_RUN
        ldr   r1,=_etext              // -> ROM data start
```

```
        ldr   r2,=_data                 // -> data start
        ldr   r3,=_edata                // -> end of data
1:      cmp   r2,r3                     // check if data to move
        ldrlo r0,[r1],#4                // copy it
        strlo r0,[r2],#4
        blo   1b                        // loop until done
#endif
// Clear .bss
// ----------
        mov   r0,#0                     // get a zero
        ldr   r1,=__bss_start           // -> bss start
        ldr   r2,=__bss_end__           // -> bss end
2:      cmp   r1,r2                     // check if data to clear
        strlo r0,[r1],#4                // clear 4 bytes
        blo   2b                        // loop until done

/*
   Call C++ constructors (for objects in "global scope")
   ctor loop added by Martin Thomas 4/2005
   based on a Anglia Design example-application for ST ARM
*/

                        LDR     r0, =__ctors_start__
                        LDR     r1, =__ctors_end__
ctor_loop:
                        CMP     r0, r1
                        BEQ     ctor_end
                        LDR     r2, [r0], #4
                        STMFD   sp!, {r0-r1}
                        MOV     lr, pc
                        MOV     pc, r2
                        LDMFD   sp!, {r0-r1}
                        B               ctor_loop
ctor_end:

// Call main program: main(0)
// --------------------------
        mov   r0,#0                     // no arguments (argc = 0)
        mov   r1,r0
        mov   r2,r0
        mov   fp,r0                     // null frame pointer
        mov   r7,r0                     // null frame pointer for thumb
        ldr   r10,=main
        mov   lr,pc

/* Enter the C code, use BX instruction so as to never return */
/* use BLX (?) main if you want to use c++ destructors below */

        bx    r10                       // enter main()

/* "global object"-dtors are never called and it should not be
   needed since there is no OS to exit to. */
/* Call destructors */
#                       LDR             r0, =__dtors_start__
#                       LDR             r1, =__dtors_end__
dtor_loop:
#                       CMP             r0, r1
#                       BEQ             dtor_end
#                       LDR             r2, [r0], #4
#                       STMFD   sp!, {r0-r1}
#                       MOV             lr, pc
#                       MOV             pc, r2
#                       LDMFD   sp!, {r0-r1}
#                       B               dtor_loop
dtor_end:
```

```
        .size   _start, . - _start
        .endfunc

        .global _reset, reset, exit, abort
        .func   _reset
_reset:
reset:
exit:
abort:
#if 0
// Disable interrupts, then force a hardware reset by driving P23 low
// ---------------------------------------------------------------------
        mrs   r0,cpsr                   // get PSR
        orr   r0,r0,#I_BIT|F_BIT        // disable IRQ and FIQ
        msr   cpsr,r0                   // set up status register

        ldr   r1,=(PS_BASE)             // PS Base Address
        ldr   r0,=(PS_PIO)              // PIO Module
        str   r0,[r1,#PS_PCER_OFF]      // enable its clock
        ldr   r1,=(PIO_BASE)            // PIO Base Address
        ldr   r0,=(1<<23)               // P23
        str   r0,[r1,#PIO_PER_OFF]      // make sure pin is contolled by PIO
        str   r0,[r1,#PIO_CODR_OFF]     // set the pin low
        str   r0,[r1,#PIO_OER_OFF]      // make it an output
#endif
        b     .                         // loop until reset

        .size _reset, . - _reset
        .endfunc

        .end
```

LPC2129-ROM.ld

```
/*****************************************************************************/
/*                                                                           */
/*  ROM.ld:  Linker Script File                                              */
/*                  "RAMfunc" demo!!                                         */
/*****************************************************************************/
ENTRY(_boot)
STACK_SIZE = 0x400;

/* Memory Definitions */
/* lpc2129 mt */
MEMORY
{
  ROM (rx) : ORIGIN = 0x00000000, LENGTH = 0x0003E000
  RAM (rw) : ORIGIN = 0x40000000, LENGTH = 0x00004000
}

/* Section Definitions */
SECTIONS
{
  /* first section is .text which is used for code */
   .text :
  {
            KEEP(*(.init))            /* Startup code from .init-section */
            *(.text .text.*)          /* remaining code */
            *(.gnu.linkonce.t.*)
            *(.glue_7)
            *(.glue_7t)
            *(.gcc_except_table)
            *(.rodata)                /* read-only data (constants) */
            *(.rodata*)
            *(.gnu.linkonce.r.*)
```

```
} > ROM

/***** old:
.text :
{
  *crt0.o (.text)
  *(.text)
  *(.rodata)
  *(.rodata*)
  *(.glue_7)
  *(.glue_7t)
} > ROM
*****/

. = ALIGN(4);

        /* .ctors .dtors are used for c++ constructors/destructors */
        /* added by Martin Thomas 4/2005 based on Anglia Design example */
        .ctors :
        {
                PROVIDE(__ctors_start__ = .);
                KEEP(*(SORT(.ctors.*)))
                KEEP(*(.ctors))
                PROVIDE(__ctors_end__ = .);
        } >ROM

        .dtors :
        {
                PROVIDE(__dtors_start__ = .);
                KEEP(*(SORT(.dtors.*)))
                KEEP(*(.dtors))
                PROVIDE(__dtors_end__ = .);
        } >ROM

  . = ALIGN(4);
        /* mthomas - end */

_etext = . ;
PROVIDE (etext = .);

/* .data section which is used for initialized data */
.data : AT (_etext)
{
  _data = .;
  *(.data)
        *(.data.*)
        *(.gnu.linkonce.d*)
        SORT(CONSTRUCTORS) /* mt 4/2005 */
        . = ALIGN(4);
        *(.fastrun)
} > RAM

. = ALIGN(4);
_edata = . ;
PROVIDE (edata = .);

/* .bss section which is used for uninitialized data */
.bss (NOLOAD) :
{
  __bss_start = . ;
  __bss_start__ = . ;
  *(.bss)
        *(.gnu.linkonce.b*)
  *(COMMON)
  . = ALIGN(4);
} > RAM
```

```
 . = ALIGN(4);
 __bss_end__ = . ;
 PROVIDE (__bss_end = .);

 .stack ALIGN(256) :
 {
   . += STACK_SIZE;
   PROVIDE (_stack = .);
 } > RAM

 _end = . ;
 PROVIDE (end = .);

 /* Stabs debugging sections.  */
 .stab          0 : { *(.stab) }
 .stabstr       0 : { *(.stabstr) }
 .stab.excl     0 : { *(.stab.excl) }
 .stab.exclstr  0 : { *(.stab.exclstr) }
 .stab.index    0 : { *(.stab.index) }
 .stab.indexstr 0 : { *(.stab.indexstr) }
 .comment       0 : { *(.comment) }
 /* DWARF debug sections.
    Symbols in the DWARF debugging sections are relative to the beginning
    of the section so we begin them at 0.  */
 /* DWARF 1 */
 .debug          0 : { *(.debug) }
 .line           0 : { *(.line) }
 /* GNU DWARF 1 extensions */
 .debug_srcinfo  0 : { *(.debug_srcinfo) }
 .debug_sfnames  0 : { *(.debug_sfnames) }
 /* DWARF 1.1 and DWARF 2 */
 .debug_aranges  0 : { *(.debug_aranges) }
 .debug_pubnames 0 : { *(.debug_pubnames) }
 /* DWARF 2 */
 .debug_info     0 : { *(.debug_info .gnu.linkonce.wi.*) }
 .debug_abbrev   0 : { *(.debug_abbrev) }
 .debug_line     0 : { *(.debug_line) }
 .debug_frame    0 : { *(.debug_frame) }
 .debug_str      0 : { *(.debug_str) }
 .debug_loc      0 : { *(.debug_loc) }
 .debug_macinfo  0 : { *(.debug_macinfo) }
 /* SGI/MIPS DWARF 2 extensions */
 .debug_weaknames 0 : { *(.debug_weaknames) }
 .debug_funcnames 0 : { *(.debug_funcnames) }
 .debug_typenames 0 : { *(.debug_typenames) }
 .debug_varnames  0 : { *(.debug_varnames) }
}
```

ΚΕΝΤΡΙΚΟΣ ΚΟΜΒΟΣ

can.c

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface definitions for can.c
 * Copyright 2007, Alexandros Papanastasatos
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#include "types.h"
#include "LPC21xx.h"
#include "can.h"

/* The messages used in the application are:
 *          0x0: Dummy message for initialization
 *          0x1: Move To Floor (sent by Central-node, 1 byte)
 *          0x2: Stop (sent by Central-node, no data)
 *          0x3: Request C1GSR (sent by Central-node, no data)
 *          0x4: Request Elevator Status (sent by Central-node, no data)
 *          0x5: C1GSR (sent by Elevator-node, 4 bytes)
 *          0x6: Elevator Status (sent by Elevator-node, 1 byte)
 *          - Bits 0,1,2:Floor, Bit 3:Load, Bit 4:Stopped, Bit 5:Alarm, Bits 6,7:Target
 */

/* According to the data contained in the Erratasheet (2006 May 17) this
 * application will not use FullCAN mode, powerdown mode, CAN sleep mode or
 * the triple buffer. Only buffer 1 will be used for transmission.
 * All messages will be sent using the Self Reception Request instead of the
 * Transmission Request command (see CAN.7 in Erratasheet).
 */

/*****************************************************************************
 *
 * Function Name: canInit()
 *
 * Description:
 *    This function initializes the CAN Interface (no interrupts used)
 *
 * Calling Sequence:
 *    can_port - CAN Interface to init (1, 2, 3 or 4) (Only 1 currently available)
 *    can_btr - CAN baud rate
 *
 * Returns:
 *    1 if initialization successful, else zero
 *
 *****************************************************************************/
int canInit(uint8_t can_port, uint32_t can_btr)
{
  volatile uint32_t *pointer;
  int p;

  if ((can_port<1) || (can_port>1)) return 0; // Illegal port value
  if ((can_btr!=CANBitrate125k_12MHz) && (can_btr!=CANBitrate250k_12MHz))  return 0; // Illegal baud rate
  C1MOD = 0x00000001; // Enter Reset Mode
  C1GSR = 0x00000000; // Clear Status Register
  C1BTR = can_btr; // Set baud rate
  C1IER = 0x00000000; // Disable interrupts
```

```
 PINSEL1 |= 0x00040000; // Enable pins for selected CAN interface

 // Acceptance Filter is configured to accept messages with ID 0x5,0x6
 // According to Erratasheet (2006 May 17-CAN.6) two disabled dummy IDs
 // should be added to the last LUT address (0xE00387FC) and two more
 // at the end of the Standard Frame Format section
 AFMR = 0x00000003; // Acceptance Filter is OFF
 pointer = &CANAF_RAM_START;
 p = 0;
 SFF_sa = p;
 *pointer = 0x20052006; // Enable 0x1 and 0x2 for CAN1
 pointer++;
 p += 4;
 *pointer = 0xF7FFF7FF; // Two disabled dummy IDs at the end of SFF
 p += 4;
 SFF_GRP_sa = p;
 EFF_sa = p;
 EFF_GRP_sa = p;
 ENDofTable = p;
 pointer = &CANAF_RAM_END;
 *pointer = 0xF7FFF7FF; // Two disabled dummy IDs at the end of CAN AF RAM
 AFMR = 0x00000000; // Acceptance Filter is ON

 C1MOD = 0x00000000; // Enter normal operating mode

 // According to Erratasheet (2006 May 17-CAN.5), before beginning normal
 // operating mode, a dummy message with ID 0x0 should be transmitted
 // by setting both the Self Reception Request bit and the Abort
 // Transmission bit in the Command register simultaneously. This
 // means that the message will only be sent once.
 C1TFI1 = 0x00000000;
 C1TID1 = 0x00000000;
 C1TDA1 = 0x00000000;
 C1TDB1 = 0x00000000;
 C1CMR = 0x00000032;
 return 1;
}

/*****************************************************************************
 *
 * Function Name: canSend()
 *
 * Description:
 *    This function sends a message through a CAN Interface
 *
 * Calling Sequence:
 *    can_port - CAN Interface to use (1, 2, 3 or 4) (Only 1 currently available)
 *    msg - Pointer to the CAN message to send
 *
 * Returns:
 *    1 if message successfully sent, else zero
 *
 *****************************************************************************/
int canSend(uint8_t can_port, CAN_MSG *msg)
{
 if ((can_port<1) || (can_port>1)) return 0; // Illegal port value
 if ((C1GSR & 0x00000004)==0) return 0; // TBS is 0-transmit buffer not available
 C1TFI1 = msg->Frame;
 C1TID1 = msg->MsgID;
 C1TDA1 = msg->DatA;
 C1TDB1 = msg->DatB;
 C1CMR |= 0x00000030;
 return 1;
}
```

```
/*****************************************************************************
 *
 * Function Name: canReceive()
 *
 * Description:
 *    This function receives a CAN message through a CAN Interface
 *
 * Calling Sequence:
 *    can_port - CAN Interface to use (1, 2, 3 or 4) (Only 1 currently available)
 *    msg - Pointer to the CAN message received
 *
 * Returns:
 *    1 if message successfully received, else zero
 *
 *****************************************************************************/
int canReceive(uint8_t can_port, CAN_MSG *msg)
{
  if ((can_port<1) || (can_port>1)) return 0; // Illegal port value
  if ((C1GSR & 0x00000001)==0) return 0; // RBS is 0-no message received
  msg->Frame = C1RFS;
  msg->MsgID = C1RID;
  msg->DatA = C1RDA;
  msg->DatB = C1RDB;
  C1CMR |= 0x00000004; //Release Receive Buffer
  return 1;
}
```

main.c

```
/* This program controls the Central-node. It uses the CAN interface 1 to communicate
with the Elevator-node.
Input: (ON = 0)
        Pushbutton GND is connected to P0.2, button 1ST to P0.3 and button 2ND to P0.4.
The messages used in the application are:
        0x0: Dummy message for initialization
        0x1: Move To Floor (sent by Central-node, 1 byte)
        0x2: Stop (sent by Central-node, no data)
        0x3: Request C1GSR (sent by Central-node, no data)
        0x4: Request Elevator Status (sent by Central-node, no data)
        0x5: C1GSR (sent by Elevator-node, 4 bytes)
        0x6: Elevator Status (sent by Elevator-node, 1 byte)
        - Bits 0,1,2:Floor, Bit 3:Load, Bit 4:Stopped, Bit 5:Alarm, Bits 6,7:Target
Floor = 0,1,2,3 or 4, Target = 0,1 or 2
The Central-node listens to the UART0 and responds to the user input through the
UART as follows:
        If the user presses 'm' the node asks for the destination floor and accepts 0, 1 or 2
                then the message "Move To Floor" is sent
        If the user presses 'h' the message "Stop" is sent
        If the user presses 's' the message "Request C1GSR" is sent
        If the user presses 'e' the message "Request Elevator Status" is sent
        If the user presses 'u' the node prints its C1GSR
        If the node receives the message "Elevator Status", the status of
                the elevator is printed in the UART
        If the node receives the message "C1GSR", the C1GSR of the
                Elevator-node is printed in the UART */

#include "types.h"
#include "LPC21xx.h"
#include "config.h"
#include "armVIC.h"
#include "uart.h"
#include "sysTime.h"
#include "can.h"

/*****************************************************************************
```

```
 *
 * Function Name: lowInit()
 *
 * Description:
 *    This function starts up the PLL then sets up the GPIO pins before
 *    waiting for the PLL to lock.  It finally engages the PLL and
 *    returns
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 ****************************************************************************/
static void lowInit(void)
{
  // set PLL multiplier & divisor.
  // values computed from config.h
  PLLCFG = PLLCFG_MSEL | PLLCFG_PSEL;

  // enable PLL
  PLLCON = PLLCON_PLLE;
  PLLFEED = 0xAA;                    // Make it happen.  These two updates
  PLLFEED = 0x55;                    // MUST occur in sequence.

  // wait for PLL lock
  while (!(PLLSTAT & PLLSTAT_LOCK))
    continue;

  // enable & connect PLL
  PLLCON = PLLCON_PLLE | PLLCON_PLLC;
  PLLFEED = 0xAA;                    // Make it happen.  These two updates
  PLLFEED = 0x55;                    // MUST occur in sequence.

  // setup & enable the MAM
  MAMTIM = MAMTIM_CYCLES;
  MAMCR = MAMCR_FULL;

  // set the peripheral bus speed
  // value computed from config.h
  VPBDIV = VPBDIV_VALUE;             // set the peripheral bus clock speed
}

/*****************************************************************************
 *
 * Function Name: sysInit()
 *
 * Description:
 *    This function is responsible for initializing the program
 *    specific hardware
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 ****************************************************************************/
static void sysInit(void)
{
  lowInit();                         // setup clocks and processor port pins

  // set the interrupt controller defaults
#if defined(RAM_RUN)
  MEMMAP = MEMMAP_SRAM;              // map interrupt vectors space into SRAM
```

```
#elif defined(ROM_RUN)
  MEMMAP = MEMMAP_FLASH;                // map interrupt vectors space into FLASH
#else
#error RUN_MODE not defined!
#endif
  VICIntEnClear = 0xFFFFFFFF;           // clear all interrupts
  VICIntSelect = 0x00000000;            // clear all FIQ selections
  VICDefVectAddr = (uint32_t)reset;     // point unvectored IRQs to reset()

//  wdtInit();                          // initialize the watchdog timer
  uart0Init(UART_BAUD(HOST_BAUD), UART_8N1, UART_FIFO_8); // setup the UART
}

static char* my_itoa(int val) // Change number (0-999) to string
{
  static char buf[5] = {0};
  int i=3;
  if (val==0) {
            buf[3]="0"[0];
            return &buf[3];
  }
  for(;val && i;--i, val/=10)
            buf[i] = "0123456789"[val % 10];
  return &buf[i+1];
}

static void printGSR(uint32_t gsr)
{
  int errors;
  if (gsr & BIT(0)) uart0Puts("\nRBS: 1");
  else uart0Puts("\nRBS: 0");
  if ((gsr & BIT(1))>>1) uart0Puts("\nDOS: 1");
  else uart0Puts("\nDOS: 0");
  if ((gsr & BIT(2))>>2) uart0Puts("\nTBS: 1");
  else uart0Puts("\nTBS: 0");
  if ((gsr & BIT(3))>>3) uart0Puts("\nTCS: 1");
  else uart0Puts("\nTCS: 0");
  if ((gsr & BIT(4))>>4) uart0Puts("\nRS: 1");
  else uart0Puts("\nRS: 0");
  if ((gsr & BIT(5))>>5) uart0Puts("\nTS: 1");
  else uart0Puts("\nTS: 0");
  if ((gsr & BIT(6))>>6) uart0Puts("\nES: 1");
  else uart0Puts("\nES: 0");
  if ((gsr & BIT(7))>>7) uart0Puts("\nBS: 1");
  else uart0Puts("\nBS: 0");
  errors = (gsr & 0x00FF0000)>>16;
  uart0Puts("\nRx Error Counter: ");
  uart0Puts(my_itoa(errors));
  errors = (gsr & 0xFF000000)>>24;
  uart0Puts("\nTx Error Counter: ");
  uart0Puts(my_itoa(errors));
}

int main(void)
{
  int userInp, elevStatus, onStop, isEmpty, timerStarted, startTime;
  CAN_MSG msgMove, msgStop, msgReqC1GSR, msgReqElevStatus, msgReceived;
  sysInit();
#if defined(UART0_TX_INT_MODE) || defined(UART0_RX_INT_MODE)
  enableIRQ();
#endif
  PCONP = 0x0000220A; // Enable only Timer0, UART0, RTC, CAN1
  PINSEL1 = 0x00000000;
  IO0DIR = 0x00000000;
  onStop = 1;
  isEmpty = 1;
```

```
  timerStarted = 0;
  startTime = 0;
  initSysTime();
  uart0Puts("\n\nWelcome to the Central-node");
  uart0Puts("\nThe node accepts the inputs: m, h, s, e and u.");
  if (canInit(1, CANBitrate125k_12MHz)) uart0Puts("\nCAN successfully initialized!");
  else uart0Puts("\nCAN initialization failed.");
  pause(HUNDRED_MS);
  msgMove.Frame = 0x00010000;
  msgMove.MsgID = 0x00000001;
  msgMove.DatA = 0x00000000;
  msgMove.DatB = 0x00000000;
  msgStop = msgMove;
  msgStop.Frame = 0x00000000;
  msgStop.MsgID = 0x00000002;
  msgReqC1GSR = msgStop;
  msgReqC1GSR.MsgID = 0x00000003;
  msgReqElevStatus = msgStop;
  msgReqElevStatus.MsgID = 0x00000004;
  for (;;) {
    if ((userInp=uart0Getch())!=-1) {
              if (userInp=='m') {
                uart0Puts("\n\nMove To Floor: ");
                        while ((userInp=uart0Getch())==-1);
                        if (((userInp-'0')>=0) && ((userInp-'0')<=2)) {
                          uart0Puts(my_itoa(userInp-'0'));
                          msgMove.DatA = userInp-'0';
                          if(canSend(1, &msgMove)) uart0Puts("\nCAN message -Move To Floor- successfully
transmitted.");
                          else uart0Puts("\nCAN message -Move To Floor- was not transmitted.");
                        }
                        else uart0Puts("\nWrong floor number.");
              }
              else if (userInp=='h') {
                uart0Puts("\n\nStop");
                if(canSend(1, &msgStop)) uart0Puts("\nCAN message -Stop- successfully transmitted.");
                        else uart0Puts("\nCAN message -Stop- was not transmitted.");
              }
              else if (userInp=='s') {
                uart0Puts("\n\nRequest C1GSR");
                if(canSend(1, &msgReqC1GSR)) uart0Puts("\nCAN message -Request C1GSR- successfully
transmitted.");
                        else uart0Puts("\nCAN message -Request C1GSR- was not transmitted.");
              }
              else if (userInp=='e') {
                uart0Puts("\n\nRequest Elevator Status");
                if(canSend(1, &msgReqElevStatus)) uart0Puts("\nCAN message -Request Elevator Status-
successfully transmitted.");
                        else uart0Puts("\nCAN message -Request Elevator Status- was not transmitted.");
              }
              else if (userInp=='u') {
                        uart0Puts("\n\nCentral-node C1GSR:");
                printGSR(C1GSR);
              }
              else uart0Puts("\nNo Command");
            }
            if (canReceive(1, &msgReceived)) {
              switch (msgReceived.Frame & 0x000003FF) {
                        case 0:
                          uart0Puts("\n\nMessage -C1GSR- received");
                          uart0Puts("\nElevator-node C1GSR:");
                          pause(HUNDRED_MS);
                          printGSR(msgReceived.DatA);
                          break;
                        case 1:
                          elevStatus = msgReceived.DatA;
```

```
                    uart0Puts("\n\nMessage -Elevator Status- received");
                    pause(HUNDRED_MS);
                    uart0Puts("\nCurrent Elevator Status: ");
                    switch (elevStatus & (BIT(0)|BIT(1)|BIT(2))) {
                      case 0:
                              uart0Puts("\nCurrent Location: GND Floor");
                              break;
                            case 1:
                              uart0Puts("\nCurrent Location: Between GND and 1ST");
                              break;
                            case 2:
                              uart0Puts("\nCurrent Location: 1ST Floor");
                              break;
                            case 3:
                              uart0Puts("\nCurrent Location: Between 1ST and 2ND");
                              break;
                            case 4:
                              uart0Puts("\nCurrent Location: 2ND Floor");
                              break;
                      default:
                              uart0Puts("\nWrong Location-");
                              uart0Puts(my_itoa(elevStatus & (BIT(0)|BIT(1)|BIT(2))));
                    }
                    pause(HUNDRED_MS);
                    if (elevStatus & BIT(3)) {
                            uart0Puts("\nThe elevator is full");
                            isEmpty = 0;
                    }
                    else {
                            uart0Puts("\nThe elevator is empty");
                            isEmpty = 1;
                    }
                    pause(HUNDRED_MS);
                    if (elevStatus & BIT(4)) {
                            uart0Puts("\nThe elevator is stopped");
                            onStop = 1;
                    }
                    else {
                            uart0Puts("\nThe elevator is moving");
                            onStop = 0;
                    }
                    pause(HUNDRED_MS);
                    if (elevStatus & BIT(5)) uart0Puts("\nThe elevator alarm is ON");
                    else uart0Puts("\nThe elevator alarm is OFF");
                    pause(HUNDRED_MS);
                    switch ((elevStatus & (BIT(6)|BIT(7)))>>6) {
                      case 0:
                              uart0Puts("\nCurrent Destination: GND Floor");
                              break;
                            case 1:
                              uart0Puts("\nCurrent Destination: 1ST Floor");
                              break;
                            case 2:
                              uart0Puts("\nCurrent Destination: 2ND Floor");
                              break;
                      default:
                              uart0Puts("\nWrong Destination");
                              uart0Puts(my_itoa((elevStatus & (BIT(6)|BIT(7)))>>6));
                    }
            }
}
          if ((!(IO0PIN & BIT(2))) && onStop) { // GND = P0.2
            uart0Puts("\n\nElevator call: GND Floor");
            msgMove.DatA = 0;
            pause(HALF_SEC);
```

```
            if(canSend(1, &msgMove)) uart0Puts("\nCAN message -Move To Floor- successfully
transmitted.");
            else uart0Puts("\nCAN message -Move To Floor- was not transmitted.");
          }
          if ((!(IO0PIN & BIT(3))) && onStop) { // 1ST = P0.3
            uart0Puts("\n\nElevator call: 1ST Floor");
            msgMove.DatA = 1;
            pause(HALF_SEC);
            if(canSend(1, &msgMove)) uart0Puts("\nCAN message -Move To Floor- successfully
transmitted.");
            else uart0Puts("\nCAN message -Move To Floor- was not transmitted.");
          }
          if ((!(IO0PIN & BIT(4))) && onStop) { // 2ND = P0.4
            uart0Puts("\n\nElevator call: 2ND Floor");
            msgMove.DatA = 2;
            pause(HALF_SEC);
            if(canSend(1, &msgMove)) uart0Puts("\nCAN message -Move To Floor- successfully
transmitted.");
            else uart0Puts("\nCAN message -Move To Floor- was not transmitted.");
            pause(HALF_SEC);
            timerStarted = 0;
          }
          if (onStop && isEmpty) {
            if (timerStarted) {
                  if (getElapsedSysTICs(startTime) > (120 * ONE_SEC)) {
                    uart0Puts("\n\nThe elevator has been inactive for more than 2 minutes.");
                    uart0Puts("\nNow moving to GND Floor.");
                    msgMove.DatA = 0;
               if(canSend(1, &msgMove)) uart0Puts("\nCAN message -Move To Floor- successfully
transmitted.");
                    else uart0Puts("\nCAN message -Move To Floor- was not transmitted.");
                    pause(HALF_SEC);
                  }
            }
            else {
                  startTime = getSysTICs();
                  timerStarted = 1;
            }
          }
          else timerStarted = 0;
  }
  return 0;
}
```

ΚΟΜΒΟΣ ΑΝΕΛΚΥΣΤΗΡΑ

can.c

```
/*****************************************************************************
 *
 * $RCSfile: $
 * $Revision: $
 *
 * This module provides the interface definitions for can.c
 * Copyright 2007, Alexandros Papanastasatos
 * No guarantees, warrantees, or promises, implied or otherwise.
 * May be used for hobby or commercial purposes provided copyright
 * notice remains intact.
 *
 *****************************************************************************/
#include "types.h"
#include "LPC21xx.h"
#include "can.h"

/* The messages used in the application are:
 *          0x0: Dummy message for initialization
 *          0x1: Move To Floor (sent by Central-node, 1 byte)
 *          0x2: Stop (sent by Central-node, no data)
 *          0x3: Request C1GSR (sent by Central-node, no data)
 *          0x4: Request Elevator Status (sent by Central-node, no data)
 *          0x5: C1GSR (sent by Elevator-node, 4 bytes)
 *          0x6: Elevator Status (sent by Elevator-node, 1 byte)
 *          - Bits 0,1,2:Floor, Bit 3:Load, Bit 4:Stopped, Bit 5:Alarm, Bits 6,7:Target
 */

/* According to the data contained in the Erratasheet (2006 May 17) this
 * application will not use FullCAN mode, powerdown mode, CAN sleep mode or
 * the triple buffer. Only buffer 1 will be used for transmission.
 * All messages will be sent using the Self Reception Request instead of the
 * Transmission Request command (see CAN.7 in Erratasheet).
 */

/*****************************************************************************
 *
 * Function Name: canInit()
 *
 * Description:
 *    This function initializes the CAN Interface (no interrupts used)
 *
 * Calling Sequence:
 *    can_port - CAN Interface to init (1, 2, 3 or 4) (Only 1 currently available)
 *    can_btr - CAN baud rate
 *
 * Returns:
 *    1 if initialization successful, else zero
 *
 *****************************************************************************/
int canInit(uint8_t can_port, uint32_t can_btr)
{
  volatile uint32_t *pointer;
  int p;

  if ((can_port<1) || (can_port>1)) return 0; // Illegal port value
  if ((can_btr!=CANBitrate125k_12MHz) && (can_btr!=CANBitrate250k_12MHz))  return 0; // Illegal baud rate
  C1MOD = 0x00000001; // Enter Reset Mode
  C1GSR = 0x00000000; // Clear Status Register
  C1BTR = can_btr; // Set baud rate
  C1IER = 0x00000000; // Disable interrupts
```

```
 PINSEL1 |= 0x00040000; // Enable pins for selected CAN interface

 // Acceptance Filter is configured to accept messages with ID 0x1,0x2,0x3,0x4
 // According to Erratasheet (2006 May 17-CAN.6) two disabled dummy IDs
 // should be added to the last LUT address (0xE00387FC) and two more
 // at the end of the Standard Frame Format section
 AFMR = 0x00000003; // Acceptance Filter is OFF
 pointer = &CANAF_RAM_START;
 p = 0;
 SFF_sa = p;
 *pointer = 0x20012002; // Enable 0x1 and 0x2 for CAN1
 pointer++;
 p += 4;
 *pointer = 0x20032004; // Enable 0x3 and 0x4 for CAN1
 pointer++;
 p += 4;
 *pointer = 0xF7FFF7FF; // Two disabled dummy IDs at the end of SFF
 p += 4;
 SFF_GRP_sa = p;
 EFF_sa = p;
 EFF_GRP_sa = p;
 ENDofTable = p;
 pointer = &CANAF_RAM_END;
 *pointer = 0xF7FFF7FF; // Two disabled dummy IDs at the end of CAN AF RAM
 AFMR = 0x00000000; // Acceptance Filter is ON

 C1MOD = 0x00000000; // Enter normal operating mode

 // According to Erratasheet (2006 May 17-CAN.5), before beginning normal
 // operating mode, a dummy message with ID 0x0 should be transmitted
 // by setting both the Self Reception Request bit and the Abort
 // Transmission bit in the Command register simultaneously. This
 // means that the message will only be sent once.
 C1TFI1 = 0x00000000;
 C1TID1 = 0x00000000;
 C1TDA1 = 0x00000000;
 C1TDB1 = 0x00000000;
 C1CMR = 0x00000032;
 return 1;
}

/*****************************************************************************
 *
 * Function Name: canSend()
 *
 * Description:
 *    This function sends a message through a CAN Interface
 *
 * Calling Sequence:
 *    can_port - CAN Interface to use (1, 2, 3 or 4) (Only 1 currently available)
 *    msg - Pointer to the CAN message to send
 *
 * Returns:
 *    1 if message successfully sent, else zero
 *
 *****************************************************************************/
int canSend(uint8_t can_port, CAN_MSG *msg)
{
 if ((can_port<1) || (can_port>1)) return 0; // Illegal port value
 if ((C1GSR & 0x00000004)==0) return 0; // TBS is 0-transmit buffer not available
 C1TFI1 = msg->Frame;
 C1TID1 = msg->MsgID;
 C1TDA1 = msg->DatA;
 C1TDB1 = msg->DatB;
 C1CMR |= 0x00000030;
```

```
  return 1;
}

/*****************************************************************************
 *
 * Function Name: canReceive()
 *
 * Description:
 *    This function receives a CAN message through a CAN Interface
 *
 * Calling Sequence:
 *    can_port - CAN Interface to use (1, 2, 3 or 4) (Only 1 currently available)
 *    msg - Pointer to the CAN message received
 *
 * Returns:
 *    1 if message successfully received, else zero
 *
 *****************************************************************************/
int canReceive(uint8_t can_port, CAN_MSG *msg)
{
  if ((can_port<1) || (can_port>1)) return 0; // Illegal port value
  if ((C1GSR & 0x00000001)==0) return 0; // RBS is 0-no message received
  msg->Frame = C1RFS;
  msg->MsgID = C1RID;
  msg->DatA = C1RDA;
  msg->DatB = C1RDB;
  C1CMR |= 0x00000004; //Release Receive Buffer
  return 1;
}
```

main.c

```
/* This program controls the Elevator-node. It uses the CAN interface 1 to communicate
            with the Central-node.
Output:
            The 7-segment display is connected to P0.4-P0.10 (a to g)
            The Direction control output is connected to P0.2 and the Step output connected to P0.3.
Input: (ON = 0)
            The Proximity sensor is connected to P0.15.
            The Load sensor is connected to P0.16.
            Pushbutton GND is connected to P0.17, button 1ST to P0.18 and button 2ND to P0.19.
            Pushbutton STOP is connected to P0.20.
            Toggle button ALARM is connected to P0.21.
The messages used in the application are:
            0x0: Dummy message for initialization
            0x1: Move To Floor (sent by Central-node, 1 byte)
            0x2: Stop (sent by Central-node, no data)
            0x3: Request C1GSR (sent by Central-node, no data)
            0x4: Request Elevator Status (sent by Central-node, no data)
            0x5: C1GSR (sent by Elevator-node, 4 bytes)
            0x6: Elevator Status (sent by Elevator-node, 1 byte)
            - Bits 0,1,2:Floor, Bit 3:Load, Bit 4:Stopped, Bit 5:Alarm, Bits 6,7:Target
Floor = 0,1,2,3 or 4, Target = 0,1 or 2 */

#include "types.h"
#include "LPC21xx.h"
#include "config.h"
#include "armVIC.h"
#include "sysTime.h"
#include "can.h"

int onStop, elevFloor, target;
uint32_t oldStatus = 0, newStatus = 0;
CAN_MSG msgElevStatus, msgC1GSR, msgReceived;
```

```
/*****************************************************************************
 *
 * Function Name: lowInit()
 *
 * Description:
 *    This function starts up the PLL then sets up the GPIO pins before
 *    waiting for the PLL to lock.  It finally engages the PLL and
 *    returns
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 *****************************************************************************/
static void lowInit(void)
{
  // set PLL multiplier & divisor.
  // values computed from config.h
  PLLCFG = PLLCFG_MSEL | PLLCFG_PSEL;

  // enable PLL
  PLLCON = PLLCON_PLLE;
  PLLFEED = 0xAA;                    // Make it happen.  These two updates
  PLLFEED = 0x55;                    // MUST occur in sequence.

  // wait for PLL lock
  while (!(PLLSTAT & PLLSTAT_LOCK))
    continue;

  // enable & connect PLL
  PLLCON = PLLCON_PLLE | PLLCON_PLLC;
  PLLFEED = 0xAA;                    // Make it happen.  These two updates
  PLLFEED = 0x55;                    // MUST occur in sequence.

  // setup & enable the MAM
  MAMTIM = MAMTIM_CYCLES;
  MAMCR = MAMCR_FULL;

  // set the peripheral bus speed
  // value computed from config.h
  VPBDIV = VPBDIV_VALUE;             // set the peripheral bus clock speed
}

/*****************************************************************************
 *
 * Function Name: sysInit()
 *
 * Description:
 *    This function is responsible for initializing the program
 *    specific hardware
 *
 * Calling Sequence:
 *    void
 *
 * Returns:
 *    void
 *
 *****************************************************************************/
static void sysInit(void)
{
  lowInit();                         // setup clocks and processor port pins

  // set the interrupt controller defaults
#if defined(RAM_RUN)
```

```
  MEMMAP = MEMMAP_SRAM;                // map interrupt vectors space into SRAM
#elif defined(ROM_RUN)
  MEMMAP = MEMMAP_FLASH;               // map interrupt vectors space into FLASH
#else
#error RUN_MODE not defined!
#endif
  VICIntEnClear = 0xFFFFFFFF;          // clear all interrupts
  VICIntSelect = 0x00000000;           // clear all FIQ selections
  VICDefVectAddr = (uint32_t)reset;    // point unvectored IRQs to reset()

//  wdtInit();                         // initialize the watchdog timer
}

static void updateDisplay(void)
{
        switch (elevFloor) {
                case 0: // display=0
                        IO0CLR = BIT(4)|BIT(5)|BIT(6)|BIT(7)|BIT(8)|BIT(9)|BIT(10);
                        IO0SET = BIT(4)|BIT(5)|BIT(6)|BIT(7)|BIT(8)|BIT(9);
                        break;
                case 2: // display=1
                        IO0CLR = BIT(4)|BIT(5)|BIT(6)|BIT(7)|BIT(8)|BIT(9)|BIT(10);
                        IO0SET = BIT(5)|BIT(6);
                        break;
                case 4: // display=2
                        IO0CLR = BIT(4)|BIT(5)|BIT(6)|BIT(7)|BIT(8)|BIT(9)|BIT(10);
                        IO0SET = BIT(4)|BIT(5)|BIT(7)|BIT(8)|BIT(10);
                        break;
        }
        return;
}

static int statusEvaluation(void)
{
        int temp = 0;

        temp = temp | (elevFloor & (BIT(0)|BIT(1)|BIT(2))); // Bits 0,1,2:Floor
        if (!(IO0PIN & BIT(16))) temp = temp | BIT(3); // Bit 3:Load
        else temp = temp & (~BIT(3));
        if (onStop) temp = temp | BIT(4); // Bit 4:Stopped
        else temp = temp & (~BIT(4));
        if (!(IO0PIN & BIT(21))) temp = temp | BIT(5); // Bit 5:Alarm
        else temp = temp & (~BIT(5));
        temp = temp | ((target & (BIT(0)|BIT(1)))<<6); // Bits 6,7:Target
        return temp;
}

static int keypadEvaluation(void) // Return the number of floor button pushed else -1
{
        if (!(IO0PIN & BIT(17))) return 0; // GND = P0.17
        if (!(IO0PIN & BIT(18))) return 1; // 1ST = P0.18
        if (!(IO0PIN & BIT(19))) return 2; // 2ND = P0.19
        return -1;
}

static void checkCAN(void)
{
        if (canReceive(1, &msgReceived))
                switch (msgReceived.Frame & 0x000003FF) {
                        case 0: // Move To Floor
                                if ((msgReceived.DatA<<1) != elevFloor) {
                                        target = msgReceived.DatA;
                                        onStop = 0; // Move
                                        if ((target<<1) > elevFloor) {
                                                if ((elevFloor == 1) || (elevFloor == 3)) ;
                                                else elevFloor++;
```

```
                                        }
                                        else {
                                                if ((elevFloor == 1) || (elevFloor == 3)) ;
                                                else elevFloor--;
                                        }
                                }
                                break;
                        case 1: // Stop
                                onStop = 1;
                                break;
                        case 2: // Request C1GSR
                                msgC1GSR.DatA = C1GSR;
                                canSend(1, &msgC1GSR);
                                break;
                        case 3: // Request Elevator Status
                                msgElevStatus.DatA = statusEvaluation();
                                canSend(1, &msgElevStatus);
                                break;
                }
        return;
}

static void isStopped(void)
{
        updateDisplay();
        if ((!(IO0PIN & BIT(16))) && (IO0PIN & BIT(21)) && (keypadEvaluation() > -1))
                if ((keypadEvaluation()<<1) != elevFloor) { // Move
                        target = keypadEvaluation();
                        onStop = 0;
                        if ((target<<1) > elevFloor) {
                                if ((elevFloor == 1) || (elevFloor == 3)) ;
                                else elevFloor++;
                        }
                        else {
                        if ((elevFloor == 1) || (elevFloor == 3)) ;
                                else elevFloor--;
                        }
                        return;
                }
        return;
}

static void isMoving(void)
{
        if ((target<<1) > elevFloor)
                IO0SET = BIT(2); // Direction: Up
        else
                IO0CLR = BIT(2); // Direction: Down
        while (!(IO0PIN & BIT(15))) {           // Check proximity sensor until
                pause(FIFTY_MS);                                        // the elevator has left the floor
                IO0SET = BIT(3);
                IO0CLR = BIT(3); // Move one step
        }
        pause(FIFTY_MS);
        IO0SET = BIT(3);
        IO0CLR = BIT(3); // Move one step
        if (!(IO0PIN & BIT(15))) {                              // Check proximity sensor if the
                if ((target<<1) > elevFloor) elevFloor++;       // elevator has reached the floor
                else if ((target<<1) < elevFloor) elevFloor--;
                updateDisplay();
                if ((target<<1) == elevFloor) onStop = 1;
                else if ((target<<1) > elevFloor) elevFloor++;
                else if ((target<<1) < elevFloor) elevFloor--;
        }
        return;
}
```

```
int main(void)
{
	sysInit();
	PCONP = 0x00002202; // Enable only Timer0, RTC and CAN1
	PINSEL1 = 0x00000000;
	IO0DIR = BIT(2)|BIT(3)|BIT(4)|BIT(5)|BIT(6)|BIT(7)|BIT(8)|BIT(9)|BIT(10); // 1=output
	IO0CLR = BIT(2)|BIT(3)|BIT(4)|BIT(5)|BIT(6)|BIT(7)|BIT(8)|BIT(9)|BIT(10);
	IO0SET = BIT(4)|BIT(7)|BIT(10); // display:3 lines
	initSysTime();
	pause(ONE_SEC);
	canInit(1, CANBitrate125k_12MHz);
	pause(HUNDRED_MS);
	onStop = 1;
	while (IO0PIN & BIT (15)); // Check proximity sensor that the elevator is on GND
	elevFloor = 0;
	target = 0;
	updateDisplay();
	newStatus = statusEvaluation();
	oldStatus = newStatus;
	msgElevStatus.Frame = 0x00010000;
	msgElevStatus.MsgID = 0x00000006;
	msgElevStatus.DatA = newStatus;
	msgElevStatus.DatB = 0x00000000;
	msgC1GSR = msgElevStatus;
	msgC1GSR.Frame = 0x00040000;
	msgC1GSR.MsgID = 0x00000005;
	msgC1GSR.DatA = C1GSR;
	canSend(1, &msgElevStatus);
	for (;;) {
		newStatus = statusEvaluation();
		if (newStatus != oldStatus) {
			oldStatus = newStatus;
			msgElevStatus.DatA = newStatus;
			canSend(1, &msgElevStatus);
		}
		if (!(IO0PIN & BIT(20))) onStop = 1;
		if (onStop) isStopped();
		else isMoving();
		checkCAN();
	}
  return 0;
}
```