ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΙΧΝΕΥΣΗ ΓΕΓΟΝΟΤΩΝ ΒΗΜΑΤΙΣΜΟΥ ΜΕ ΧΡΗΣΗ ΕΠΙΤΑΧΥΝΣΙΟΜΕΤΡΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ι. Ζδράγκας

Επιβλέπων : Ιωάννης Ν. Αβαριτσιώτης
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΙΧΝΕΥΣΗ ΓΕΓΟΝΟΤΩΝ ΒΗΜΑΤΙΣΜΟΥ ΜΕ ΧΡΗΣΗ ΕΠΙΤΑΧΥΝΣΙΟΜΕΤΡΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ι. Ζδράγκας

Επιβλέπων : Ιωάννης Ν. Αβαριτσιώτης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Μαίου 2008.

.....	.....	.....
Ιωάννης Ν. Αβαριτσιώτης	Ελευθέριος Καγιάφας	Βασίλειος Λούμος
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2008

.....
Γεώργιος Ι. Ζδράγκας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής εργασίας είναι η σχεδίαση και η κατασκευή ενός συστήματος μέτρησης των επιταχύνσεων που δημιουργούνται στο πόδι κατά τον βηματισμό του ανθρώπου. Με κατάλληλη επεξεργασία των μετρήσεων γίνεται η αναγνώριση των γεγονότων στον κύκλο βηματισμού και η αναγνώριση των βημάτων. Μεγάλη σημασία δίνεται σε τεχνικές αυτοματοποιημένης αναγνώρισης.

Το σύστημα αποτελείται απο δύο πλακέτες. Στη μία πλακέτα βρίσκεται ο μικροελεγκτής και το σύστημα τροφοδοσίας και στη δεύτερη ένα επιταχυνσιόμετρο τριών αξόνων και ένας αναλογικοψηφιακός μετατροπέας. Το επιταχυνσιόμετρο έχει τρεις εξόδους που οι τάσεις τους είναι ανάλογες των επιταχύνσεων στους τρεις ορθογώνιες άξονες. Ο μετατροπέας μετατρέπει τις τάσεις που λαμβάνει από το επιταχυνσιόμετρο σε ψηφιακές τιμές και τις στέλνει στον μικροελεγκτή μέσω της διασύνδεσης I2C. Ο μικροελεγκτής τις αποθηκεύει και αργότερα τις στέλνει στον υπολογιστή μέσω της σειριακής θύρας για ανάλυση.

Η επεξεργασία των μετρήσεων γίνεται με το πρόγραμμα Matlab. Έχει γραφτεί συνάρτηση η οποία επεξεργάζεται τα αρχικά σήματα, και δημιουργεί κάποια καινούργια από τα οποία μπορούμε να αντλήσουμε εύκολα πληροφορίες. Επίσης γίνεται σύγκριση μεταξύ των βηματισμών τριών ανθρώπων.

Το σύστημα είναι μικρό σε μέγεθος, εύχρηστο, αξιόπιστο, χαμηλής κατανάλωσης και πολύ χρήσιμο για μετρήσεις επιτάχυνσεων τριών αξόνων όχι μόνο στο πόδι αλλά σε οποιοδήποτε σημείο του σώματος.

Λέξεις κλειδιά
Επιταχυνσιόμετρο, Βηματισμός, Ανάλυση Βηματισμού, Μικροελεγκτής ARM, LPC2148, MMA7260.

Abstract

The purpose of this thesis is the study, design and implemenation of an integrated system for measuring the accelerations produced by the foot during the gait. With proper measurement processing we can identify the gait events during a gait cycle and the number of footsteps.

The system consists of two printed circuit boards. The first one is the pcb with the microcontroller and the power supply unit while in the second one the accelerometer and the analog to digital converter take place. The accelerometer has three outputs, whose voltages are proportional to the acceleration based on the three orthogonal axes. The analog to digital converter converts the signals from the accelerometer into digital values and sends them to the microcontroller through the I2C bus. The microcontroller stores these values in order to send them later to the main computer through the serial port. The computer analyses the data and produces the results.

Matlab is used for processing the data. A function created for processing the initial signals returns new ones,from which useful information can be easily derived.

The system is small in size , easy to use, reliable, low power and very useful for three axes acceleration measurements , not only on foot but on many places of the body as well.

Keywords
Accelerometer, Gait, Gait Analysis, ARM Microcontroller, LPC2148, MMA7260.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας, κύριο Ιωάννη Αβαριτσιώτη για την ευκαιρία που μου έδωσε να εργαστώ κοντά του, για τις γνώσεις που μου μετέδωσε και την βοήθεια που μου προσέφερε κατά τη διάρκεια της διπλωματικής μου εργασίας. Επίσης,ευχαριστώ τους κυρίους Ιωάννη Θανασόπουλο, Γεώργιο Παπανικολάου και Απόστολο Ψαρρό για την βοήθειά τους στην κατασκευή των ηλεκτρονικών πλακετών και στη συλλογή των δεδομένων.

Τέλος, θέλω να ευχαριστήσω τους φίλους μου, την αδερφή μου Δήμητρα και την Λουίζα για την συμπαράστασή τους και για την μεγάλη βοήθεια που μου προσέφεραν.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Ο βηματισμός είναι ξεχωριστό χαρακτηριστικό για κάθε άνθρωπο και εξαρτάται από διάφορους παράγοντες όπως το βάρος του, το μήκος των άκρων του, τα υποδήματά του, τη στάση του, την ηλικία του, το φύλλο του, τη διάθεσή του και την χαρακτηριστική κίνηση που εκτελεί. Μπορούμε να επεκτείνουμε τον ορισμό του τρόπου βηματισμού όχι μόνο στο τρόπο κίνησης των ποδιών του ανθρώπου αλλά σε όλη την κίνηση του, όπως την κίνηση των χεριών, τις γωνίες που σχηματίζουν, την ταχύτητα των άκρων, την περίοδο και την συχνότητα βηματισμού και την κίνηση του κορμού.

Η ανάλυση βηματισμού μπορεί να αντλήσει στοιχεία για τον εξεταζόμενο, να ταξινομήσει τους ανθρώπους με βάση κάποιο ιδιαίτερο χαρακτηριστικό τους και να αναγνωρίσει μέχρι ποιος περπατάει μέσα από κάποιο δείγμα ανθρώπων που ξέρουμε από πριν το χαρακτηριστικό βηματισμό τους. Συγκεκριμένα μπορούμε με την ανάλυση βηματισμού να βρούμε χαρακτηριστικά στον τρόπο περπατήματος όπως:

1) Πότε ξεκινάει και πότε σταματάει ένας άνθρωπος την κίνησή του
2) Την κατεύθυνσή του
3) Την σταθερότητα στον βηματισμό του
4) Την προσαρμοστικότητα του ανθρώπου στις αλλαγές του εδάφους
5) Το ρυθμό του βηματισμού
6) Την ταχύτητα κίνησης
7) Το μήκος του κάθε βήματος
8) Τη μέση τιμή του μήκους των βημάτων
9) Την επιτάχυνση στα σημεία που θέλουμε
10) Την επιτάχυνση στην κίνησή του
11) Κινητικά προβλήματα (αν υπάρχουν)
12) Το φύλλο του ανθρώπου

Χαρακτηριστικά του εδάφους που περπατάει όπως:

1) Την κλίση του εδάφους που περπατάει ο άνθρωπος
2) Την ολισθηρότητα του εδάφους

Υπάρχουν λοιπόν πολλά στοιχεία που μπορούμε να βρούμε από την ανάλυση του βηματισμού και που μπορούν να χρησιμοποιηθούν για ιατρικούς σκοπούς όπως αποκατάσταση ασθενών μετά από τραυματισμό ή εγχείρηση, στον αθλητισμό, στην εκπαίδευση και στην έρευνα.

Στην παρούσα διπλωματική γίνεται η κατασκευή ενός συστήματος λήψης δεδομένων με ένα επιταχυνσιόμετρο τριών αξόνων και στην συνέχεια η ανάλυση τους. Παρουσιάζουμε αλγόριθμους για την αυτοματοποιημένη ανίχνευση χαρακτηριστικών γεγονότων στον βηματισμό καθώς επίσης και τον αριθμό των βημάτων που εκτελούνται. Τέλος συγκρίνουμε τον βηματισμό διαφόρων ανθρώπων για να βρούμε τις ομοιότητες και τις διαφορές τους.

ΚΕΦΑΛΑΙΟ 2

Τεχνολογίες μελέτης βηματισμού

2.1 Εισαγωγή

Η ανάλυση βηματισμού είναι η μελέτη της κίνησης των ανθρώπων και ζώων κατά τη διάρκεια του βηματισμού τους. Βρίσκει πολλές εφαρμογές στην ιατρική για αντιμετώπιση τραυματισμών, κινητικών προβλημάτων και διαταραχών καθώς επίσης στον αθλητισμό για να πετυχαίνουν οι αθλητές αποδοτικότερη και ασφαλέστερη κίνηση.

Η ιστορία της ανάλυσης βηματισμού αρχίζει με την ανάπτυξη της φωτογραφίας. Με διαδοχικές φωτογραφίες μπορούσε να απεικονιστεί η κίνηση των ανθρώπων και να μελετηθούν κινητικά προβλήματα. Η εφαρμογή της ανάλυσης βηματισμού για ιατρικούς σκοπούς έδωσε τεράστια ώθηση στην έρευνα και στην δημιουργία νέων τεχνικών.

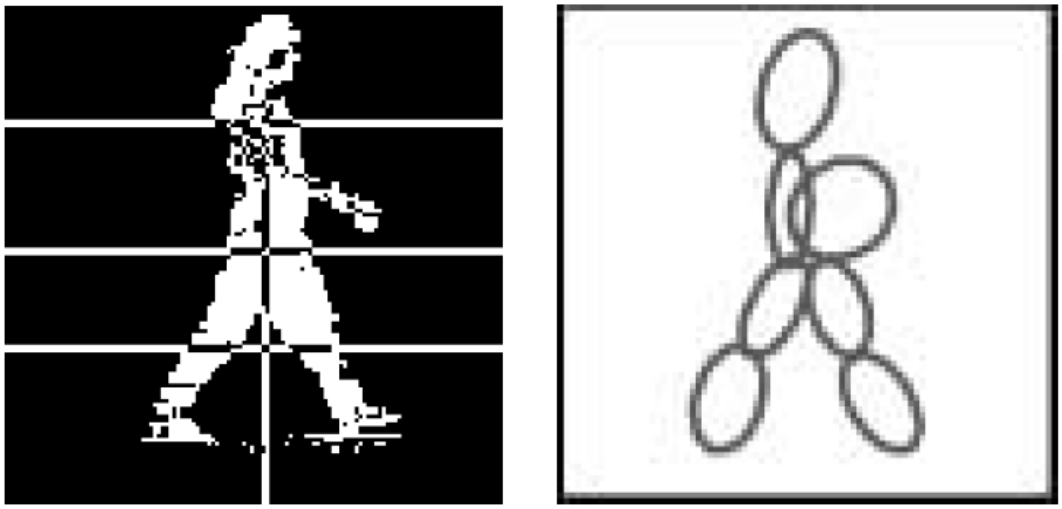
2.2 Τεχνικές ανάλυσης βηματισμού

2.2.1 Ανάλυση με βίντεο

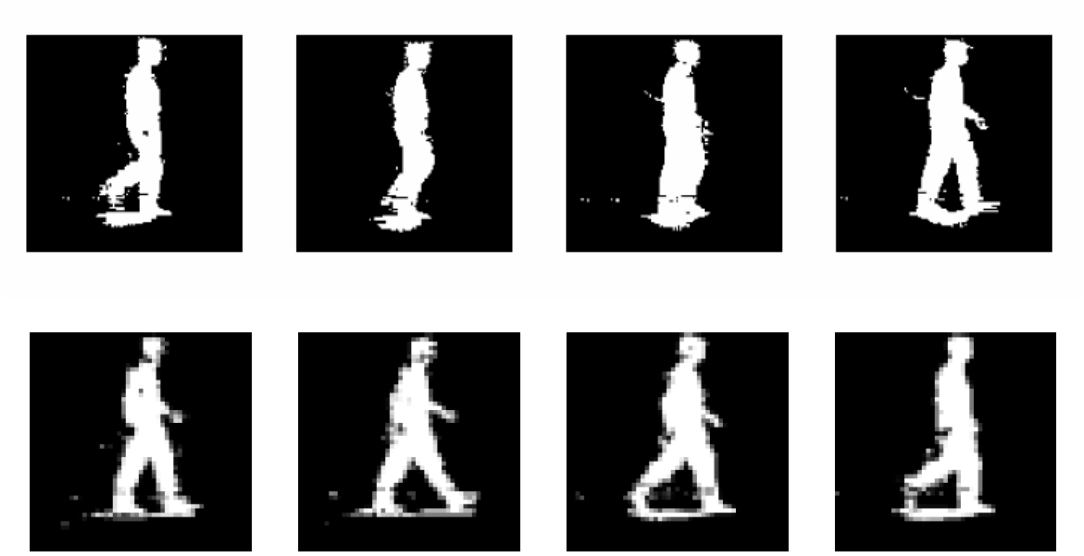
Μια από τις πιο διαδεδομένες τεχνικές ανάλυσης βηματισμού είναι η ανάλυση με βίντεο. Με αυτή την τεχνική γίνεται καταγραφή της κίνησης που θέλουμε να μελετήσουμε με χρήση βιντεοκάμερας και στη συνέχεια μπορούμε να την προβάλουμε σε πιο αργή ταχύτητα ώστε να παρατηρήσουμε τα χαρακτηριστικά που μας ενδιαφέρουν καθώς επίσης να καταλάβουμε της αλληλουχία κινήσεων που χρειάζονται για την εκτέλεση μιας συγκεκριμένης διαδικασίας κίνησης. Ένας πιο αυτοματοποιημένος τρόπος γίνεται με την χρήση ηλεκτρονικών υπολογιστών και την υλοποίηση αλγορίθμων. Ένα παράδειγμα αλγόριθμου είναι η απόμονωση του ανθρώπου από το περιβάλλον, ο διαχωρισμός του σε οκτώ μέρη (κεφάλι, κορμός, δεξί χέρι, αριστερό χέρι, και από δύο μέρη το κάθε πόδι) και στη συνέχεια η ανάλυση της κίνησης με βάση αυτά τα μέρη, όπως οι γωνίες που σχηματίζουν μεταξύ τους, η αλληλουχία των κινήσεών τους και η ταχύτητά τους. Επιπλέον μπορούν να χρησιμοποιηθούν πολλαπλές κάμερες για να παρατηρηθεί η κίνηση από διαφορετικές οπτικές γωνίες. Η τεχνική αυτή μπορεί να δώσει χρήσιμα αποτελέσματα για τη μέτρηση γωνιών των αρθρώσεων και της ταχύτητας και βοηθάει στην τρισδιάστατη απεικόνιση κινήσεων.

Έχει μελετηθεί ότι για την αναγνώριση ανθρώπων από τον τρόπο βηματισμού τους, αρκούν οι γωνίες που σχηματίζουν οι αρθρώσεις τους. Η τεχνική αυτή

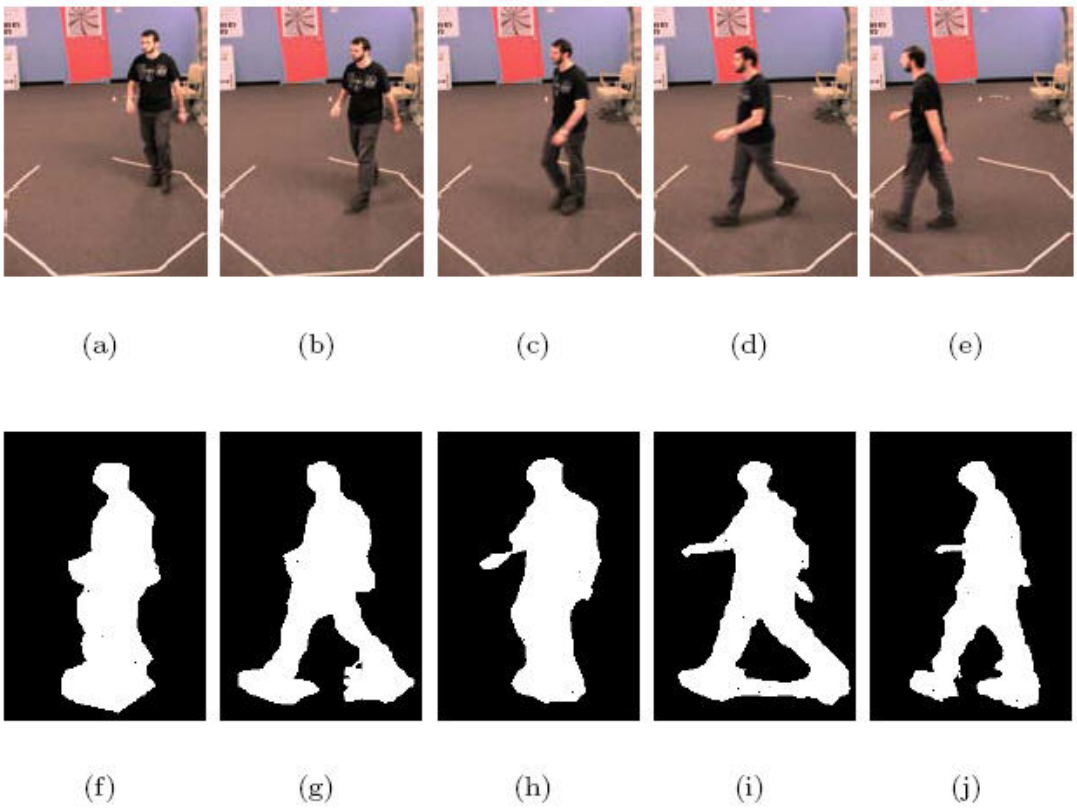
δεν απαιτεί την χρήση υψηλής ανάλυσης βίντεο. Ωστόσο η κυριότερη δυσκολία είναι ο διαχωρισμός του ανθρώπου από το υπόλοιπο περιβάλλον, ο ρουχισμός που μπορεί να εμποδίζει την καθαρή εμφάνιση των αρθρώσεων, όταν ο άνθρωπος έχει μαζί του κάποιο μεγάλο αντικείμενο όπως μια τσάντα και όταν η κίνηση γίνεται σε μη αποδοτικές γωνίες λήψης. Το τελευταίο πρόβλημα αντιμετωπίζεται με την χρήση πολλαπλών βιντεοκαμερών.



Σχήμα 2-1: Η απομονωμένη από το περιβάλλον σιλουέτα ενός ανθρώπου και ο χωρισμός της σε επτά μέρη, κάθε μέρος έχει αντικατασταθεί από μία έλλειψη.

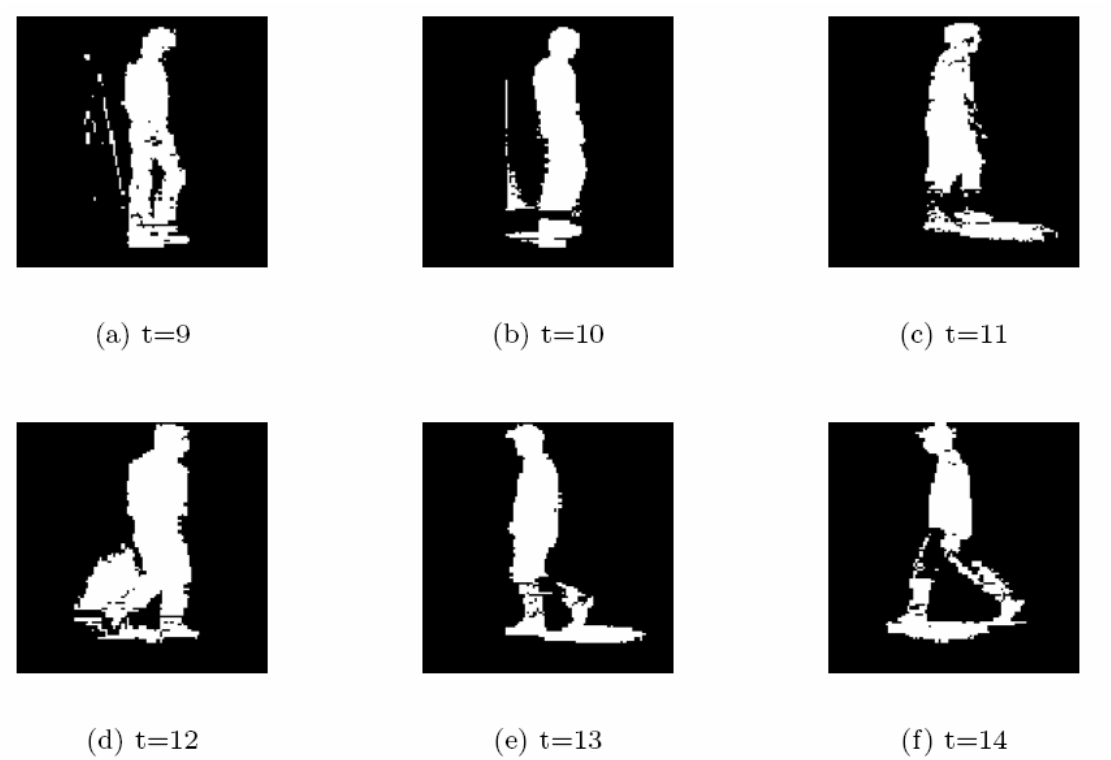


Σχήμα 2-2: Ο κύκλος βηματισμού ενός ανθρώπου που έχει απομονωθεί γραφικά από το υπόλοιπο περιβάλλον.



Σχήμα 2-3: Ο βηματισμός ενός ανθρώπου σε γωνίες που δημιουργούν λάθη στην απεικόνηση της σιλουέτας του.

Σχήμα 2-4: Σιλουέτες με διαφορετικά ρούχα και χτενίσματα και οι διαφορές στην απεικόνησή τους.

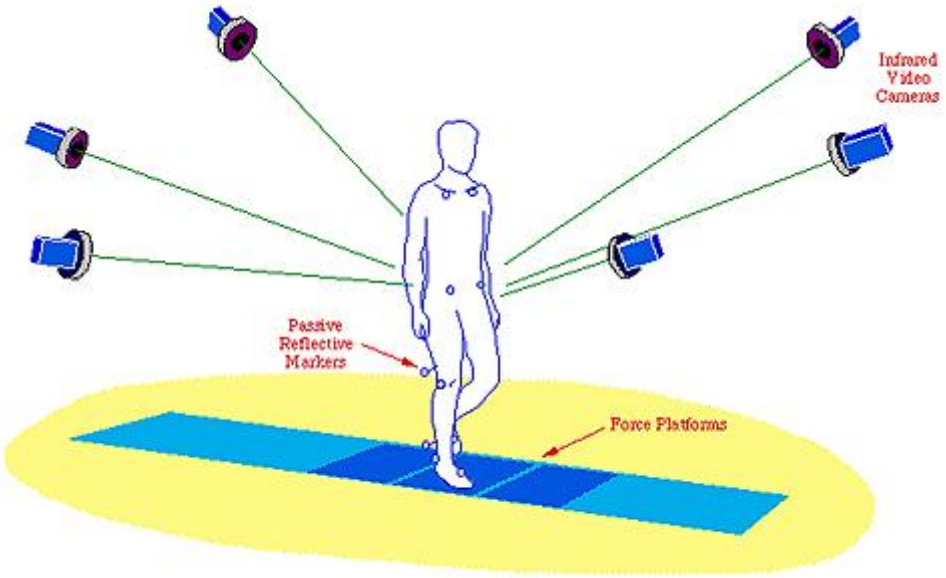


Σχήμα 2-5: Σιλουέτες που δεν έχουν απομονωθεί σωστά από το περιβάλλον.

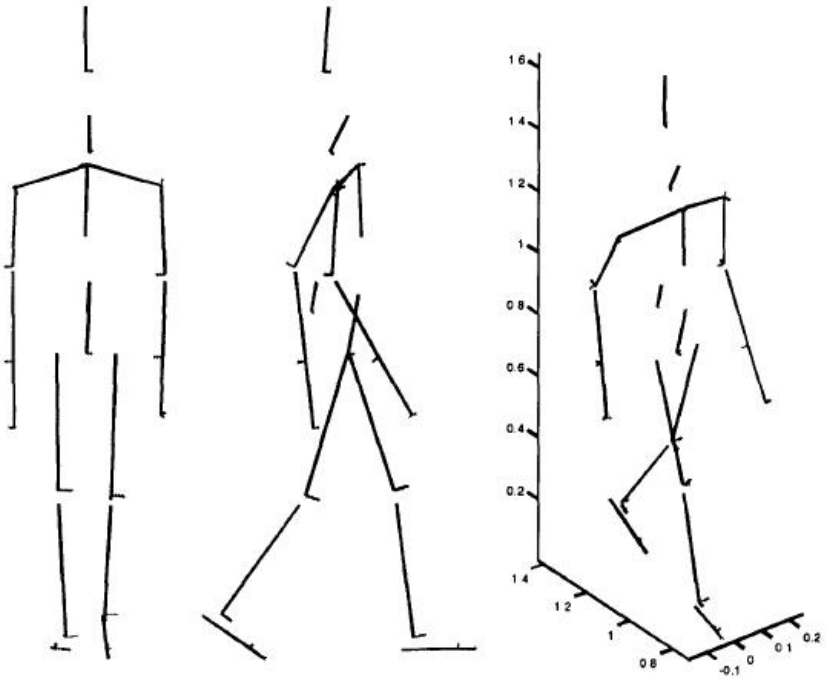
2.2.2 Παθητικά συστήματα ανίχνευσης με ανακλαστήρες

Η τεχνική αυτή χρησιμοποιεί μικρές ανακλαστικές επιφάνειες που τοποθετούνται σε κατάλληλα σημεία στο σώμα του ανθρώπου (πχ. αρθρώσεις και άκρα) και κάμερες. Συνήθως χρησιμοποιούνται μέχρι οκτώ κάμερες, οι οποίες στέλνουν υπέρυθρα σήματα και ανιχνεύουν την ανάκλαση από τις ανακλαστικές επιφάνειες. Η τεχνική βασίζεται στην γωνία και στην χρονική καθυστέρηση μεταξύ του αρχικού σήματος και του ανακλώμενου. Με τις τιμές αυτές και με την τεχνική του τριγωνισμού γίνεται πολύ ακριβής εντοπισμός του ανακλαστήρα και συνεπώς της κίνησης. Το σύστημα από τα δεδομένα που λαμβάνει μπορεί να σχηματίσει τρισδιάστατη αναπαράσταση της κίνησης.

Οι έρευνες έχουν δείξει ότι με αυτή την τεχνική μπορεί να γίνει ανίχνευση του φύλου του ανθρώπου που παρατηρείται και μάλιστα η ανίχνευση αυτή είναι πιο εύκολη όταν οι ανακλαστήρες είναι τοποθετημένοι στα άνω άκρα και αρθρώσεις του ανθρώπου. Η επιτυχία στην ανίχνευση του φύλλου με αυτό τον τρόπο είναι πολύ μεγαλύτερη του 65%. Το σύστημα αυτό μπορεί να εφαρμοστεί και για τη σύλληψη κίνησης για χρήση στις κινηματογραφικές ταινίες.

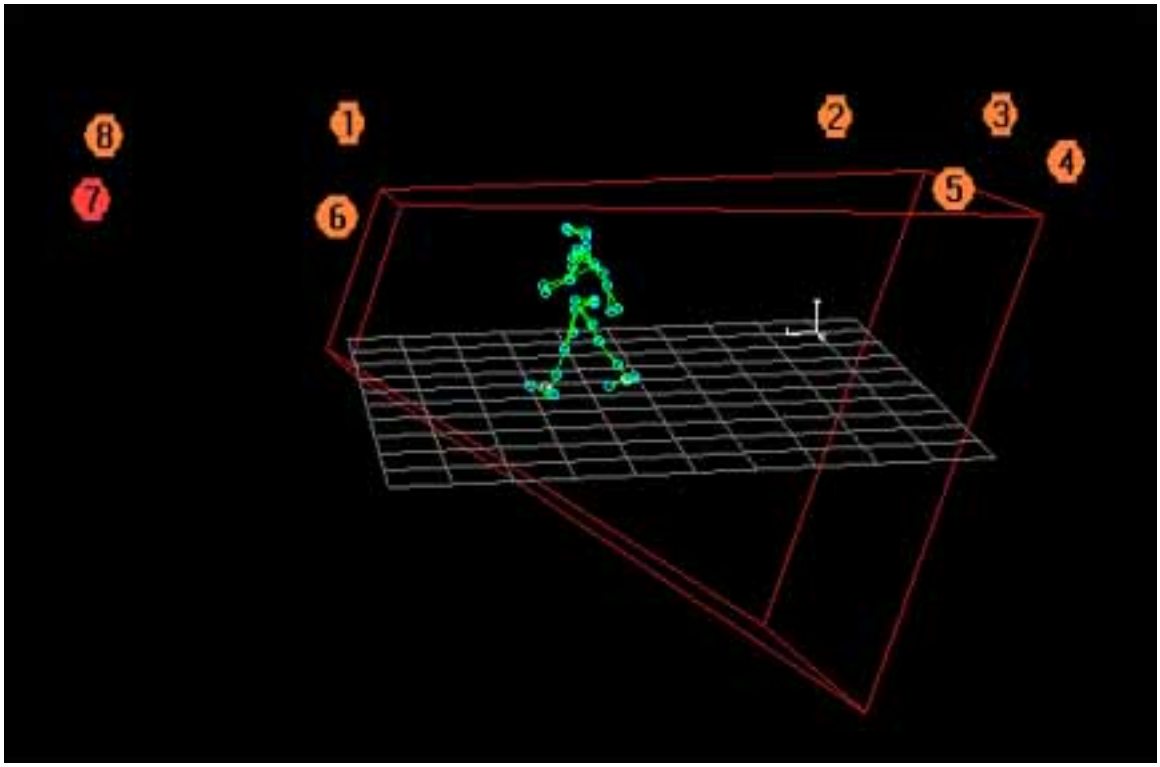


Σχήμα 2-6: Το σύστημα ανίχνευσης με παθητικούς ανακλαστήρες.



Σχήμα 2-7: Τρισδιάστατη απεικόνιση που προέρχεται από επεξεργασία δεδομένων συστήματος ανακλαστήρων.

Σχήμα 2-8: Άνθρωπος που του έχουν τοποθετηθεί ανακλαστήρες.



Σχήμα 2-9: Τρισδιάστατη απεικόνιση συστήματος ανακλαστήρων με οκτώ κάμερες.

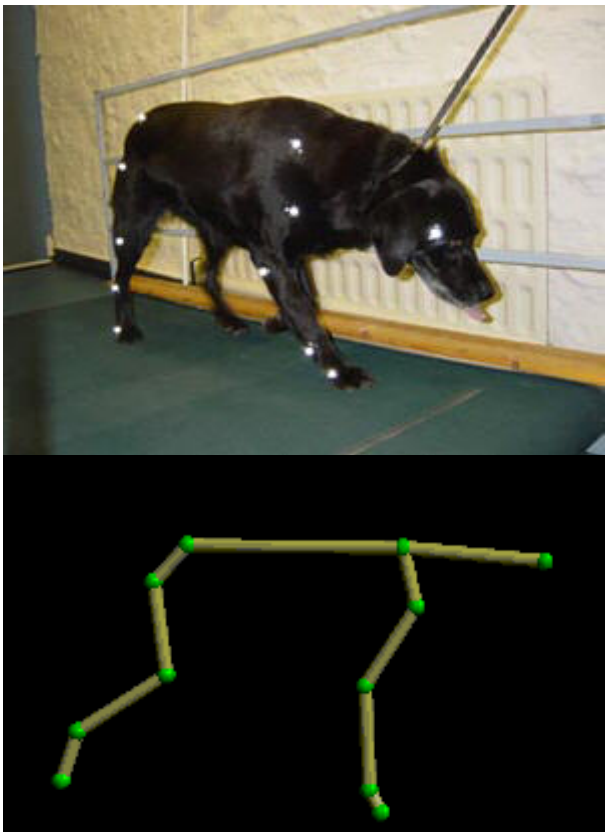
2.2.3 Ενεργά συστήματα ανίχνευσης υπερύθρων

Η τεχνική αυτή μοιάζει με αυτή των παθητικών συστημάτων ανίχνευσης αλλά χρησιμοποιεί ενεργά στοιχεία. Αυτά τοποθετούνται στο σώμα του ανθρώπου και μόλις ανιχνεύσουν το υπέρυθρο σήμα, αντιδρούν στέλνοντας ένα καινούριο σήμα. Το σήμα αυτό χρησιμοποιείται στην τεχνίκη τριγωνισμού για τον εντοπισμό της θέσης του ανακλαστήρα. Το σύστημα αυτό είναι καλύτερο από το σύστημα των παθητικών σημείων διότι κάθε σημείο ανίχνευσης έχει την δική του συχνότητα σήματος και συνεπώς τη δική του ταυτότητα. Έτσι αποφεύγονται τα λάθη συμβολής μεταξύ των σημάτων των σημείων. Ωστόσο τα συστήματα αυτά έχουν μεγαλύτερο πρόβλημα όταν τα σημεία ανίχνευσης δεν έχουν άμεση έκθεση στο υπέρυθρο σήμα.

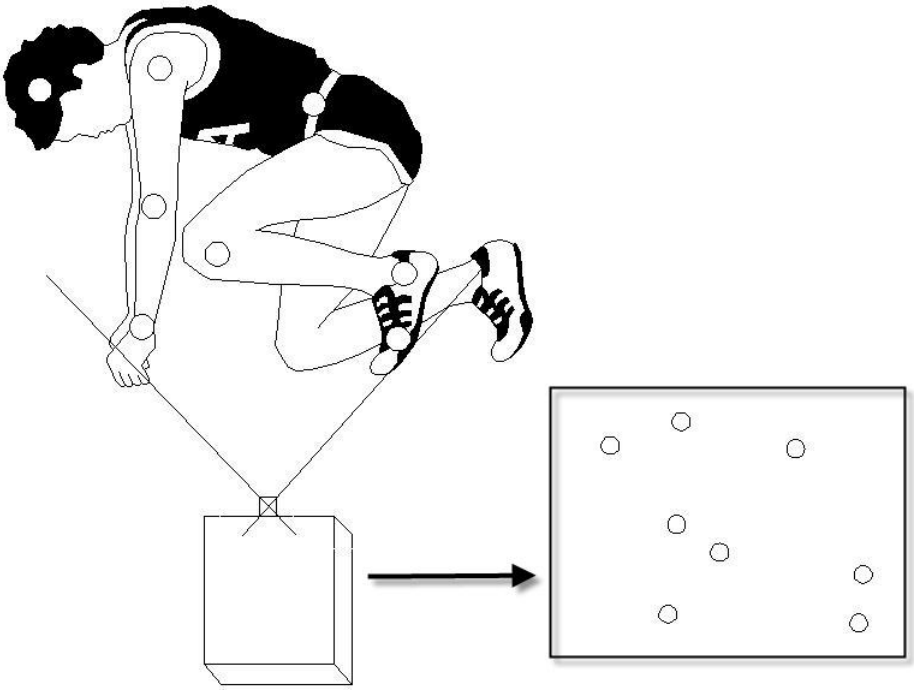
Σχήμα 2-10: Παιδί με ενεργούς ανακλαστήρες τοποθετημένους πάνω στο σώμα του.

Σχήμα 2-11: Ενήλικος με ενεργούς ανακλαστήρες τοποθετημένους στο σώμα του.

Σχήμα 2-12: Γυναίκα που βαδίζει σε διάδρομο με ανακλαστήρες στα πόδια της.



Σχήμα 2-13: Σκύλος με ανακλαστήρες και η απεικόνηση της κίνησής του.

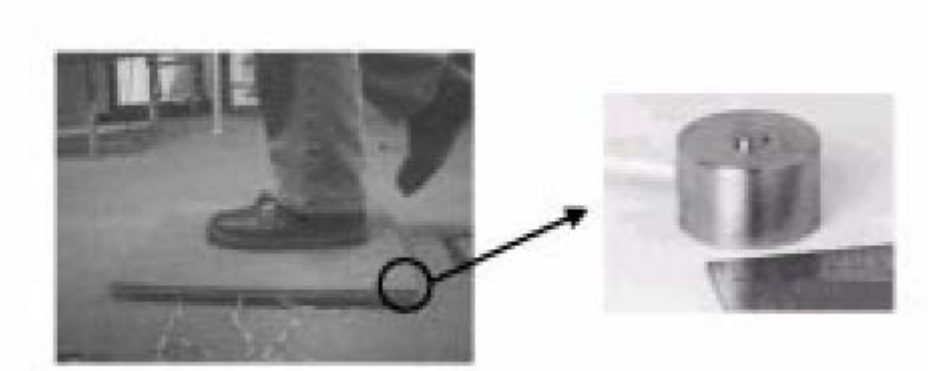


Σχήμα 2-14: Αθλητής με ανακλαστήρες και η απεικόνηση της κίνησής του.

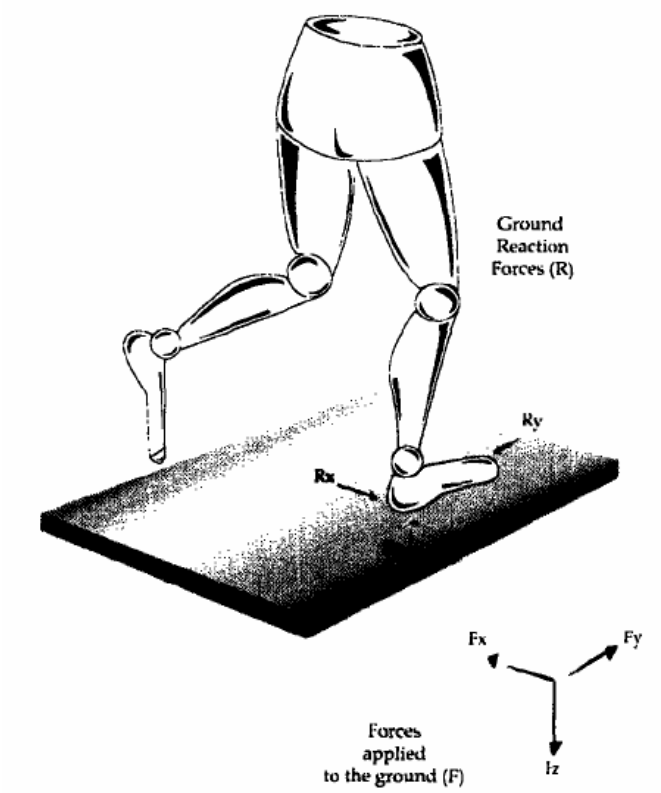
2.2.4 Πλατφόρμες μέτρησης δύναμης αντίδρασης

Η πλατφόρμα μέτρησης δύναμης είναι ένα ιατρικό σύστημα που μετράει την δύναμη που ασκείται πάνω της. Αποτελείται από δύο επίπεδα που έχουν ανάμεσά τους αισθητήρες δύναμης που είναι είτε πιεζοηλεκτρικοί είτε παραμόρφωσης (strain gauges). Το κάτω επίπεδο εφαρμόζεται σταθερά στο έδαφος ενώ στο πάνω μπορεί να περπατήσει κάποιος άνθρωπος. Η πλατφόρμα μπορεί να μετράει με ακρίβεια τις δυνάμεις που ασκέι ένας άνθρωπος στο έδαφος κατά τον βηματισμό του. Οι αισθητήρες τοποθετούνται στις γωνίες τις πλατφόρμας σε ορθογώνιο τρόπο μεταξύ τους και μπορούν να μετρούν τόσο την κατακόρυφη δύναμη όσο και την οριζόντια προς την διεύθυνση της κίνησης του ανθρώπου. Με επεξεργασία των μετρήσεων από κάθε αισθητήρα μπορεί να υπολογιστεί το σημείο άσκησης δύναμης πάνω στην πλατφόραμα καθώς και το μέγεθός της. Οι πλατφόρμες μπορούν να συνδυαστούν με άλλα συστήματα ανάλυσης κίνησης, όπως βίντεο, για καλύτερα αποτελέσματα.

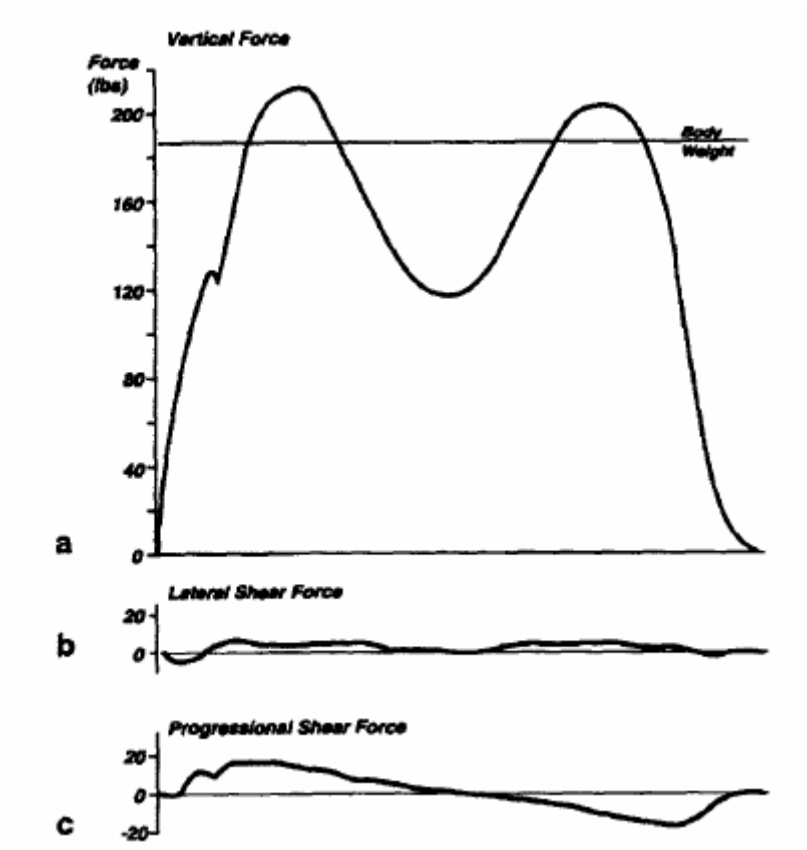
Όταν ένας άνθρωπος στέκεται ακίνητος, περπατάει ή τρέχει, υπάρχει συνεχής αλληλεπίδραση μεταξύ του ίδιου και του εδάφους. Σε αυτή την αλληλεπίδραση το βάρος του σώματος “πέφτει” στο πόδι που στηρίζει το σώμα και μεταφέρεται κατά την κίνηση του σώματος. Με αυτό τον τρόπο δημιουργούνται κάθετες και περιστροφικές δυνάμεις στο έδαφος. Σύμφωνα με τον τρίτο νόμο του Νεύτωνα, υπάρχει μια δύναμη αντίδρασης από την επιφάνεια, ίδιας έντασης αλλά αντίθετης φοράς από την δύναμη που ασκεί το πόδι στο έδαφος. Οι δυνάμεις μπορούν να μετρηθούν από την πλατφόρμα και να επεξεργαστούν στην συνέχεια. Από αυτές προκύπτουν χαρακτηριστικά διαγράμματα.



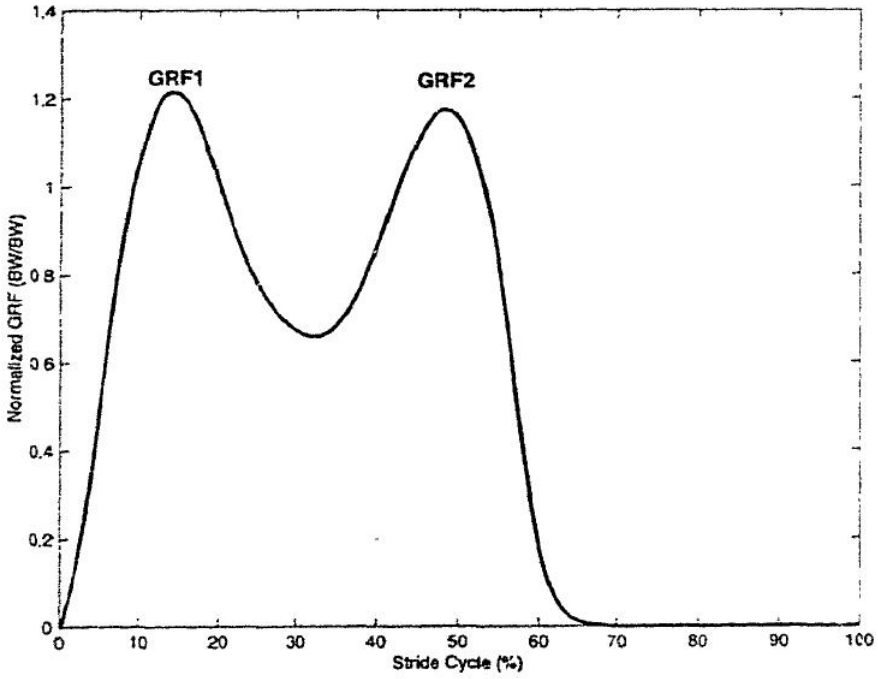
Σχήμα 2-15: Άνθρωπος που πατάει σε πλατφόρα μέτρησης δύναμης αντίδρασης και ένας από τους αισθητήρες δύναμης.



Σχήμα 2-16: Οι δυνάμεις που ασκούνται στο σημείο επαφής του ποδιού με το έδαφος.



Σχήμα 2-17: Οι καμπύλες των τριών ορθογώνιων δυνάμεων που “διαβάζει” η πλατφόρμα μέτρησης δύναμης αντίδρασης.



Σχήμα 2-18: Η κατακόρυφη δύναμη αντίδρασης του εδάφους σε συνάρτηση με το ποσοστό του βηματισμού.

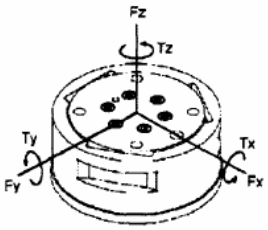
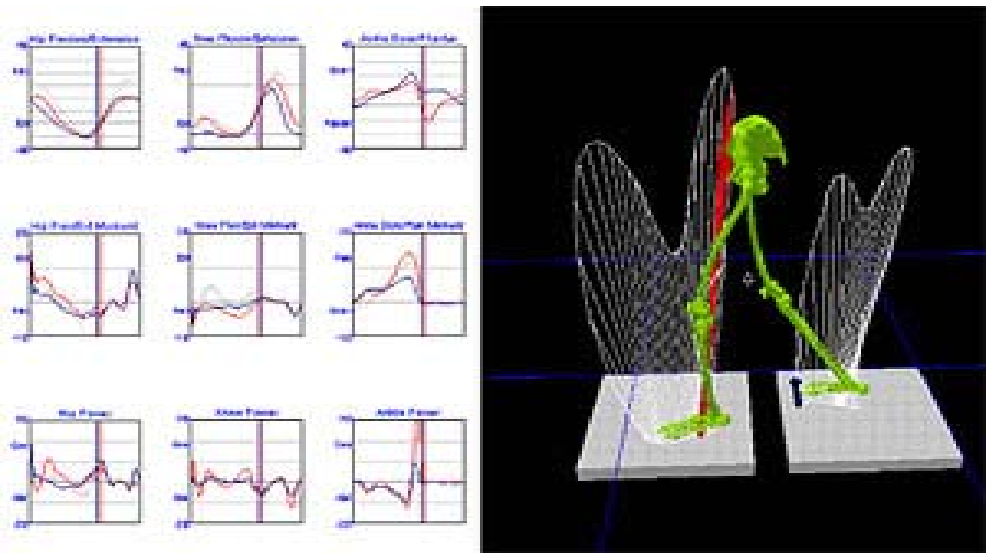


Figure 2-21. ATI Industrial Automation, Inc. force / torque transducer: *Delta*

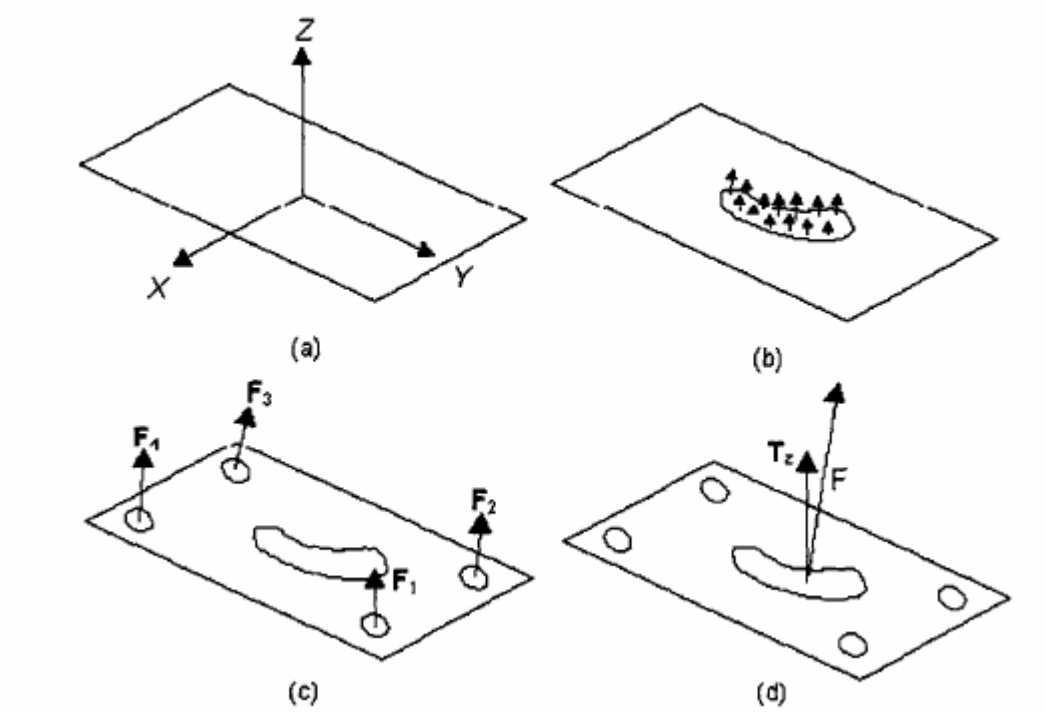
Fx,Fy	Fz	Tx,Ty	Tz	Fx,Fy	Fz	Tx,Ty	Tz
660 N	1980 N	60 N.m	60 N.m	1/32 N	1/32 N	9/533 N.m	6/533 N.m
SENSING RANGES				RESOLUTION			

Σχήμα 2-19: Ένας αισθητήρας δύναμης που χρησιμοποιείται σε πλατφόρμες μέτρησης δύναμης αντίδρασης και τα χαρακτηριστικά του.



Σχήμα 2-20: Τρισδιάστατη απεικόνιση βηματισμού με βάση τις δυνάμεις που μετρήθηκαν από δύο πλατφόρμες.

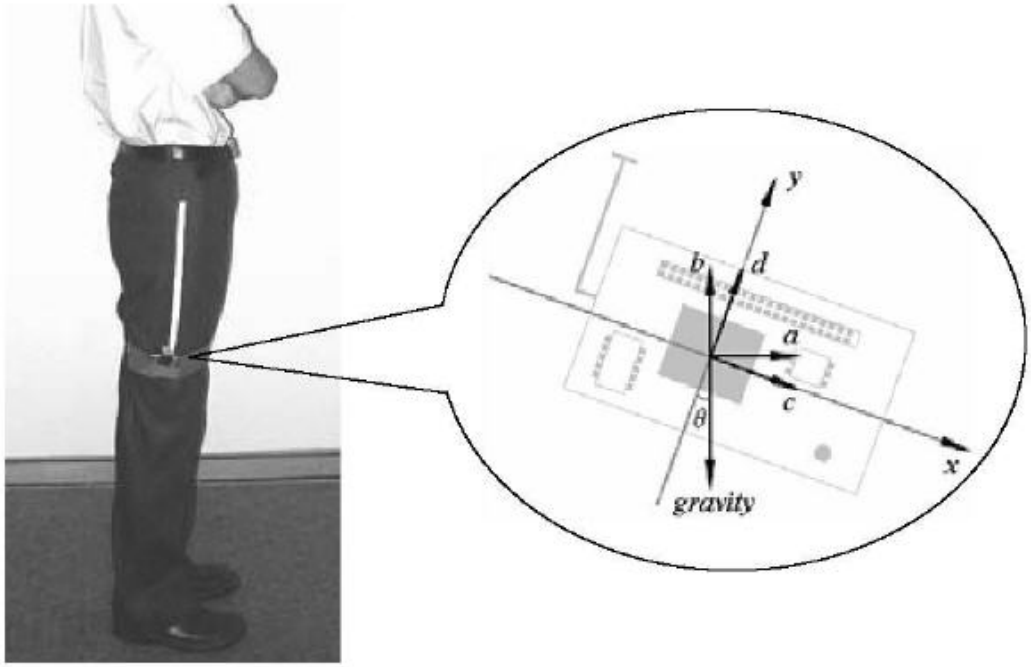
Το κέντρο πίεσης είναι το σημείο αναφοράς που επιδρούν όλες οι δυνάμεις που ασκούνται στην πλατφόρμα. Σε αυτό το σημείο μπορούμε να θεωρήσουμε ότι ασκείται η συνισταμένη δύναμη από το σώμα. Με αυτό τον τρόπο μπορούμε να απλοποιήσουμε πολύ την ανάλυση.



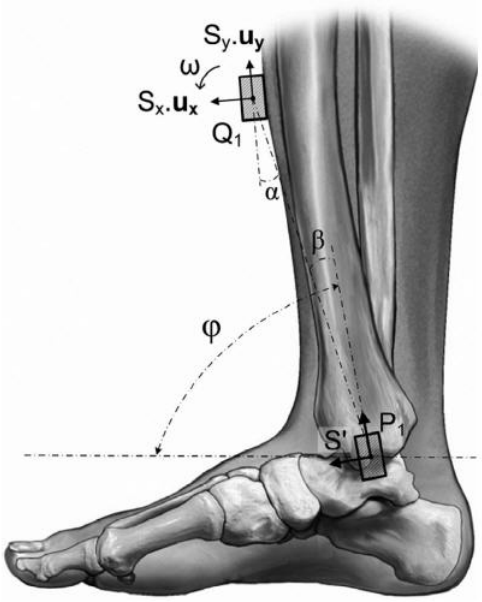
Σχήμα 2-21: Οι δυνάμεις που ασκούνται στην επιφάνεια που καταλαμβάνει το πέλμα και η συνισταμένη τους.

2.2.5 Συστήματα επιταχυνσιομέτρων

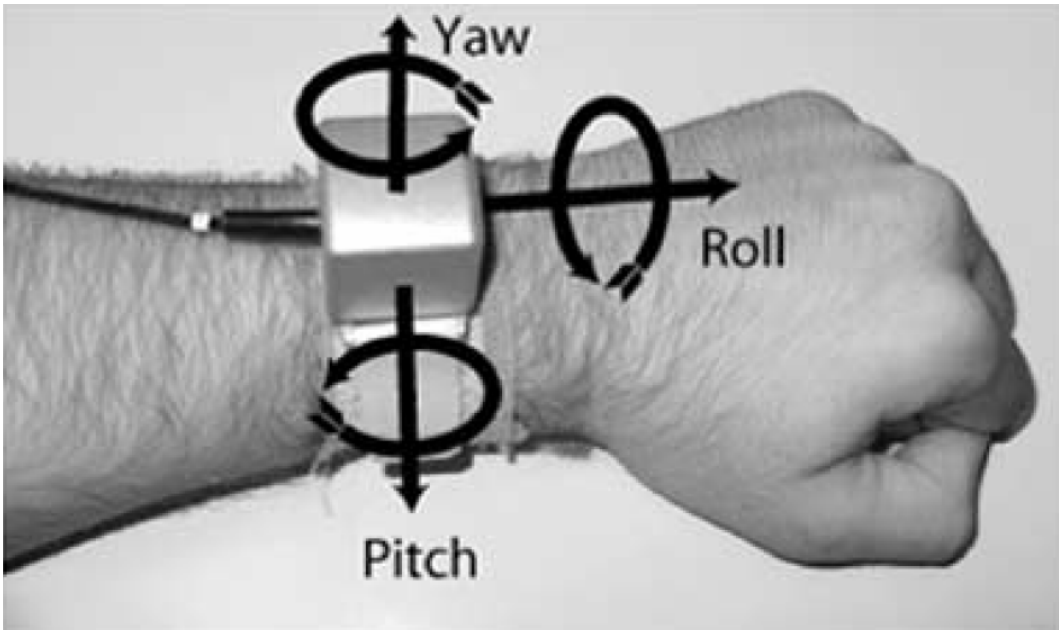
Τα συστήματα επιταχυνσιομέτρων έχουν αρχίζει να αναπτύσονται όλο και περισσότερο. Αισθητήρες επιτάχυνσης τοποθετούνται σε διαφορετικά σημεία του σώματος και μετράνε τις επιταχύνσεις του κάθε σημείου. Τα επιταχυνσιόμετρα μπορούν να μετράνε τις επιταχύνσεις από ενός μέχρι τριών αξόνων οι οποίοι είναι ορθογώνιοι μεταξύ τους. Συνήθως συνοδεύονται από έναν μικροελεγκτή ο οποίος αναλαμβάνει να στέλνει τα δεδομένα είτε ενσύρματα είτε ασύρματα σε έναν κεντρικό σταθμό βάσης στον οποίο γίνεται η επεξεργασία των δεδομένων.



Σχήμα 2-22: Άνθρωπος που του έχει τοποθετηθεί στο πόδι επιταχυνσιόμετρο τριών αξόνων.



Σχήμα 2-23: Επιταχυνσιόμετρο τοποθετημένο στο πόδι ενός ανθρώπου.



Σχήμα 2-24: Ένας αισθητήρας επιτάχυνσης για μεγάλης διάρκειας επίβλεψης ασθενών με τη νόσο του Πάρκινσον.

2.3 Συγκριση των τεχνικών ανάλυσης βηματισμού

Κάθε σύστημα ανίχνευσης και ανάλυσης βηματισμού έχει διαφορετικά χαρακτηριστικά και μπορεί να χρησιμοποιηθεί ανάλογα με τις προδιαγραφές της εφαρμογής μας.

Τα συστήματα που χρησιμοποιούν το βίντεο δεν απαιτούν την εφαρμογή αισθητήρων πάνω στον άνθρωπο και μπορούν να χρησιμοποιήσουν κάμερες χαμηλής ανάλυσης. Η δυσκολία παρουσιάζεται στο ότι πρέπει να ξεχωρίζουν τους ανθρώπους από το περιβάλλον και ότι πρέπει να υπάρχει άμεση οπτική επαφή υπό τη σωστή γωνία. Επίσης οι αλγόριθμοι που χρησιμοποιούνται είναι πιο δύσκολοι να υλοποιηθούν από εκείνους στα υπόλοιπα συστήματα. Ωστόσο μπορούν να χρησιμοποιούν τις υπάρχουσες κάμερες ασφαλείας σε έναν χώρο και συνεπώς η τεχνική αυτή μπορεί να εφαρμόζεται σε ανοιχτούς χώρους, καταστήματα και γραφεία.

Τα συστήματα με παθητικούς και ενεργούς ανακλαστήρες δίνουν πολύ καλά αποτελέσματα καταγραφής κίνησης. Οι ανακλαστήρες μπορούν να τοποθετηθούν σε όποια σημεία του σώματος θέλουμε. Τα δεδομένα που παίρνουμε μπορούν να παράγουν τρισδιάστατα βίντεο της κίνησης που περιέχουν την μέγιστη πληροφορία κίνησης που μπορούμε να πάρουμε. Ωστόσο τα συστήματα αυτά είναι τα πιο ακριβά, χρειάζονται εξειδικευμένο εξοπλισμό και η καταγραφή της κίνησης γίνεται μόνο σε ειδικό εργαστήριο αφού πρώτα τοποθετήσουμε τους ανακλαστήρες πάνω στον άνθρωπο που εξετάζουμε.

Τα συστήματα μέτρησης δύναμης αντίδρασης του εδάφους δίνουν περιορισμένα δεδομένα για τον βηματισμό καθώς δεν μετρούν καθόλου την

κίνηση όταν το πόδι είναι στον αέρα. Επιπλέον δεν μπορούν να πάρουν στοιχεία για τιν κίνηση και τις γωνίες. Επιπλέον μπορούν να μετρήσουν την δύναμη μόνο σε ένα ή δύο βήματα και όχι ολόκληρου του βηματισμού. Συνεπώς οι εφαρμογές τους είναι περιορισμένες. Τα συστήματα αυτά μπορούν να χρησιμοποιηθούν σε συνδυασμό με τα υπόλοιπα ή όταν υπάρχει περιορισμένος χώρος κίνησης. Είναι ιδανικά όταν μας ενδιαφέρει η κίνηση που σχετίζεται με τα πόδια και ειδικά για ιατρικούς λόγους όπως κινητικά προβλήματα, μέτρηση πλατυποδιάς κ.α.

Τα συστήματα με τα επιταχυνσιόμετρα είναι τα πιο πρακτικά. Είναι μικρά, μπορούν να τοποθετηθούν σε όποιο σημείο του σώματος θέλουμε, είναι φτηνά, δεν απαιτούν κάποιο εξειδικευμένο εργαστήριο για να γίνουν μετρήσεις, μπορούν να παίρνουν δεδομένα για μεγάλες χρονικές περιόδους και μπορούμε να πάρουμε ακριβείς μετρήσεις χωρίς να υπάρχει οπτική επαφή. Πρόκειται για μια τεχνολογία που αναπτύσσεται όλο και περισσότερο και γι' αυτό το λόγο επιλέχηκε αυτό το είδος συστήματος για να γίνει η ανάλυση βηματισμού σε αυτή την διπλωματική εργασία.

ΚΕΦΑΛΑΙΟ 3

Το σύστημα μέτρησης επιτάχυνσης

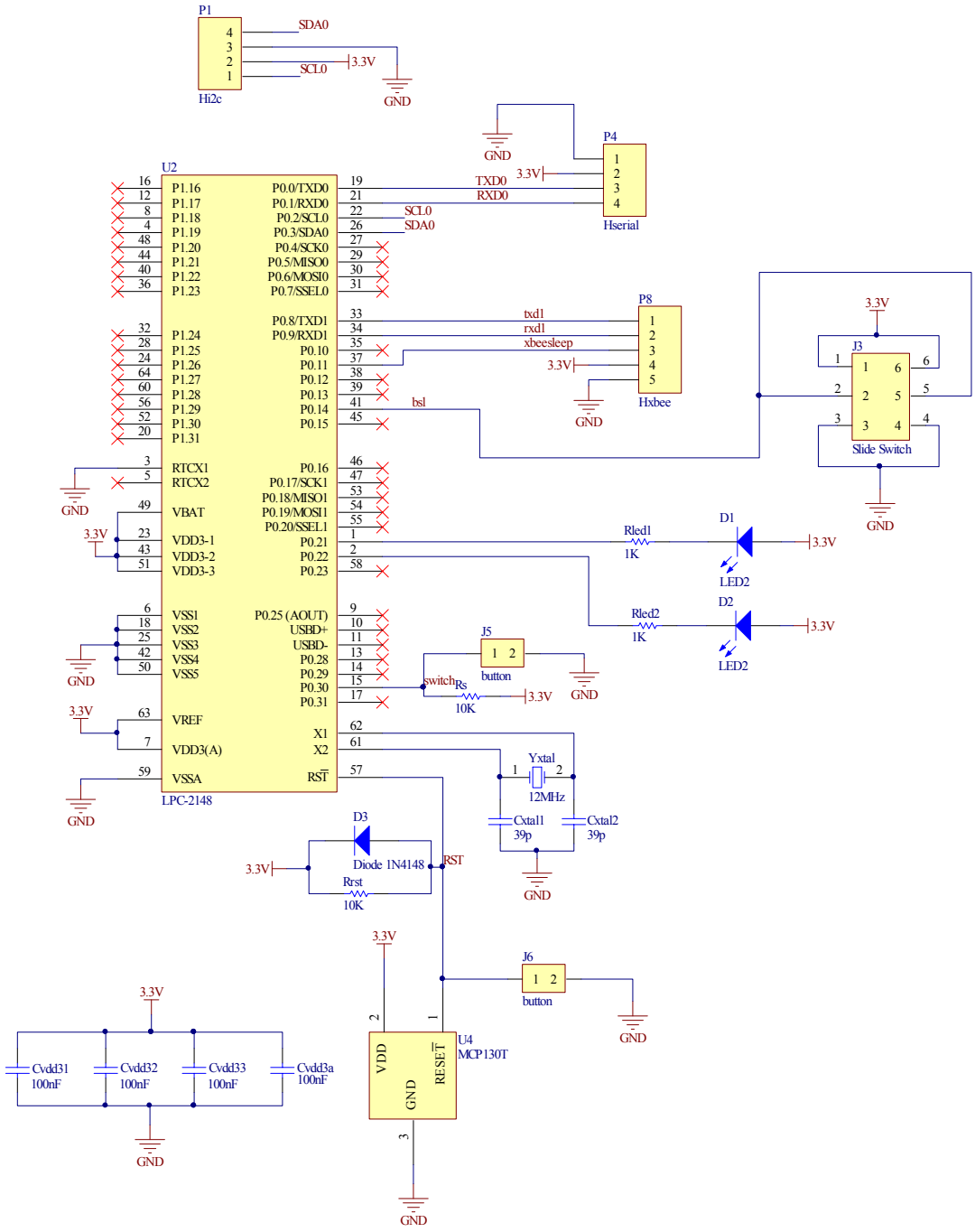
3.1 Αρχιτεκτονική του συστήματος

Το σύστημα αποτελείται από δύο πλακέτες. Στην μια πλακέτα (σχήμα 3-1) βρίσκεται ο μικροελεγκτής LPC2148 και το τροφοδοτικό. Στην άλλη (σχήμα 3-6) υπάρχει το επιταχυνσιόμετρο τριών αξόνων MMA7260 το οποίο μετατρέπει τις επιταχύνσεις σε τάση και ένας μετατροπέας αναλογικού σήματος σε ψηφιακό (ADC) MAX1237 ο οποίος επικοινωνεί με τον μικροελεγκτή μέσω του πρωτοκόλλου I2C. Αν και ο μικροελεγκτής που χρησιμοποιούμε έχει εσωτερικό ADC ακρίβειας 10 bit, χρησιμοποιήθηκε ο ADC MAX1237 επειδή έχει μεγαλύτερη ακρίβεια 12 bit αλλά κυρίως επειδή θέλαμε η μετατροπή να γίνεται στην πλακέτα του επιταχυνσιομέτρου καθώς αν μεταφέραμε τις αναλογικές εξόδους του για δειγματοληψία στην πλακέτα του μικροελεγκτή, ο θόρυβος πάνω στο σήμα θα ήταν μεγάλος. Επίσης χρησιμοποιήσαμε δύο πλακέτες ώστε να πετύχουμε μικρότερο μέγεθος στην πλακέτα με το επιταχυνσιόμετρο και για μεγαλύτερη ευκολία στην τοποθέτησή της.

3.2 Αναλυτική περιγραφή των υποσυστημάτων

3.2.1 Υποσύστημα μικροελεγκτή

Ο μικροελεγκτής έχει τον κεντρικό ρόλο στο σύστημα καθώς επικοινωνεί με τον ADC και λαμβάνει τα δεδομένα. Επιπλέον ανάβει κάποια ενδεικτικά LEDs για να μας δείχνει την κατάσταση του συστήματος. Επεξεργάζεται τα δεδομένα και τα στέλνει στον ηλεκτρονικό υπολογιστή. Το σχηματικό διάγραμμα το κυκλώματος με τον μικροελεγκτή παρουσιάζεται στο σχήμα 3-1.



Σχήμα 3-1: Σχηματικό διάγραμμα συστήματος μικροελεγκτή.

Ο μικροελεγκτής LPC2148 παρατηρούμε ότι έχει τέσσερις εισόδους τροφοδοσίας τις οποίες έχουμε συνδέσει στα 3.3V μέσω decoupling capacitors 100nF. Επιπλέον στα 3.3V έχουμε συνδέσει και το VBAT που είναι η είσοδος για την ανεξάρτητη τροφοδοσία του real time clock (RTC) του μικροελεγκτή. Αυτό το έχουμε κάνει για να καταναλώνει λιγότερη ενέργεια το LPC2148.

Για τον προγραμματισμό του μικροελεγκτή πάνω στο κύκλωμα χρησιμοποιείται το περιφερειακό του UART0, το οποίο επικοινωνεί σειριακά με τον ηλεκτρονικό υπολογιστή που έχουμε τον κώδικα που θέλουμε να

“φορτώσουμε” στον μικροελεγκτή. Όταν θέλουμε να προγραμματίσουμε το LPC2148 κάνουμε reset έχοντας το pin41-P0.14 στα 0Volt. Με αυτό τον τρόπο ο μικροελεγκτής καταλαβαίνει ότι θέλουμε να τον προγραμματίσουμε και αρχίζει να εκτελεί ένα πρόγραμμα (bootloader) που έχει καταχωρημένο στην μνήμη Flash. Στην συνέχεια στέλνουμε τον κώδικα σε μορφη “.hex” σειριακά στο UART0 μέσω του προγράμματος LPC2000 Flash Utility. Γι’ αυτό τον λόγο έχουμε στο κύκλωμα μια ακιδοσειρά που συνδέεται με το UART0 του μικροελεγκτή και έναν διακόπτη (slide switch) για να επιλέγουμε αν θέλουμε 3.3V ή 0V στο P0.14 .

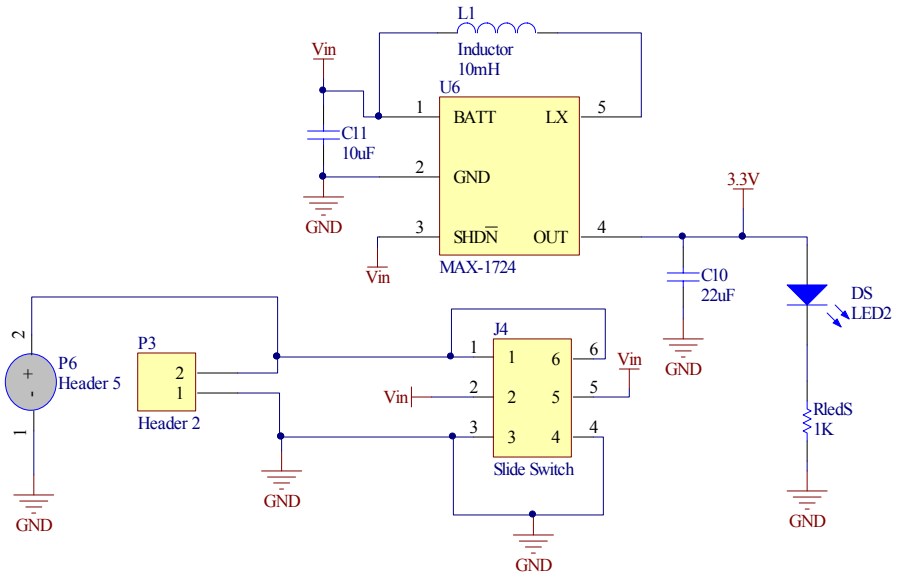
Επιπλέον έχουμε εφοδιάσει το κύκλωμα με 2 leds και έναν κουμπί. Το κουμπί χρησιμοποιείται για να ελέγχουμε πότε ο μικροελεγκτής θα αρχίσει να παίρνει τα δείγματα και πότε θα τα στείλει στον υπολογιστή. Είναι ένα απλό button switch συνδεδεμένο στα 0V και επιπλέον υπάρχει μια pull-up αντίσταση ώστε όταν δεν έχουμε πατημένο το κουμπί, ο μικροελεγκτής να έχει είσοδο 3.3V.Τα leds μας δείχνουν πότε άρχισε και πότε τελείωσε η δειγματοληψία και η απόστολή των δεδομένων στον υπολογιστή.

Η συχνότητα λειτουργίας του μικροελεγκτή καθορίζεται από τον κρύσταλλο των 12MHz. Η συχνότητα αυτή πολλαπλασιάζεται εσωτερικά στο LPC2148 μέσω του PLL και δημιουργεί τη συχνότητα λειτουργίας του μικροελεγκτή στα 60MHz. Οι τιμές των πυκνωτών επιλέχτηκαν με βάση το datasheet του LPC2148 για τον συγκεκριμένο κρύσταλλο.

Το κύκλωμα reset για τον μικροελεγκτή αποτελείται από το ολοκληρωμένο MCP130T το οποίο κάνει reset όταν η τάση τροφοδοσίας πέφτει κάτω από τα 3V. Με αυτόν τον τρόπο εξασφαλίζεται η σωστή λειτουργία του μικροελεγκτή καθώς όταν λειτουργεί κάτω από την ονομαστική τάση λειτουργίας υπάρχει κίνδυνος να αρχίσει να εκτελεί εντολές χωρίς να υπάρχει αρκετή τάση για την σωστή λειτουργία της RAM και της Flash μνήμης. Επιπλέον κρατάει τον μικροελεγκτή σε κατάσταση reset μέχρι η τάση να φτάσει σε σταθερό σημείο λειτουργίας κατά το άνοιγμα του τροφοδοτικού. Το κουμπί J6 χρησιμοποιείται για να κάνει reset τον μικροελεγκτή κάθε φορά που το πατάμε και η δίοδος D3 προστατεύει το κύκλωμα από υψηλές τάσεις. Τέλος υπάρχουν τρεις ακιδοσειρές για επικοινωνία με τα UART0, UART1 και I2C περιφερειακά του μικροελεγκτή.

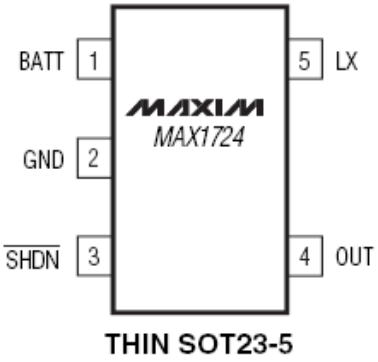
3.2.2 Υποσύστημα τροφοδοσίας

Στο σχήμα 3-2 παρουσιάζεται το σχηματικό διάγραμμα του τροφοδοτικού. Βασίζεται στο ολοκληρωμένο MAX1724 το οποίο είναι ένας Step-Up DC-DC Converter και χρησιμοποιείται για να ανυψώνει την τάση στα 3.3V από μια μικρότερη. Αυτό γίνεται για να μπορεί το σύστημα να λειτουργεί από μια μικρή μπαταρία λιθίου (Li+ cell), μία ή δύο αλκαλικές ή NiMH μπαταρίες. Το υποσύστημα έχει επίσης ένα led για να μας δείχνει πότε το κύκλωμα είναι ανοιχτό και έναν διακόπτη.



Σχήμα 3-2: Σχηματικό διάγραμμα τροφοδοτικού.

Το MAX1724 της MAXIM (σχήμα 3-3) είναι ένας Step-Up DC-DC μετατροπές, υψηλής απόδοσης, πακεταρισμένο σε SOT23-5. Έχει πολύ μικρό ρεύμα ηρεμίας 1.5μA και μπορεί να δώσει 3.3V από μόλις 0.9V. Συνεπώς είναι ιδανικό για λειτουργία από μικρές μπαταρίες όπως μια μια μπαταρία λιθίου.



Σχήμα 3-3: Το MAX1724.

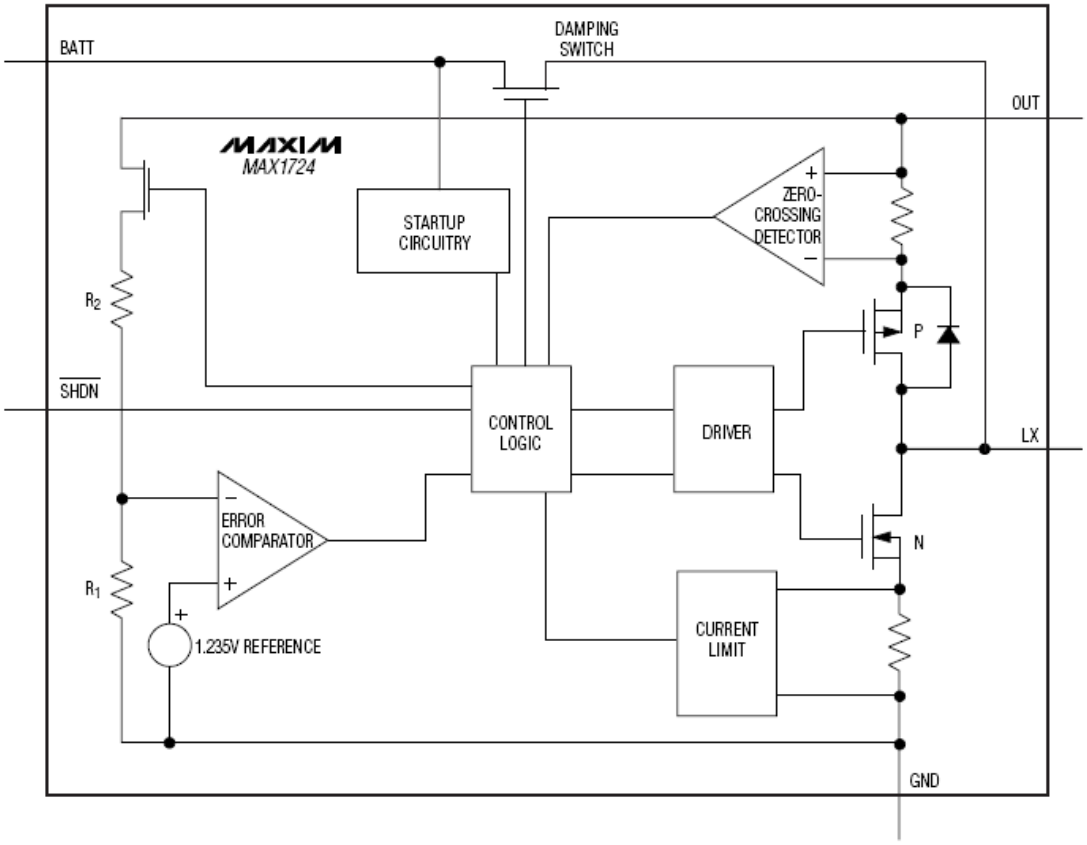
Έχει τα εξής χαρακτηριστικά:

-Μέχρι 90% απόδοση
-Δεν χρειάζεται εξωτερική δίοδο ή FET
-1.5μA ρεύμα ηρεμίας
-±1% ακρίβεια τάσης εξόδου
-Μέχρι 150mA ρεύμα εξόδου
-0.8V με 5.5V τάση εισόδου
-0.91V εγγυημένη τάση έναρξης
-Εσωτερική EMI καταστολή
-SOT23-5 συσκευασία(1.1mm μέγιστο ύψος)

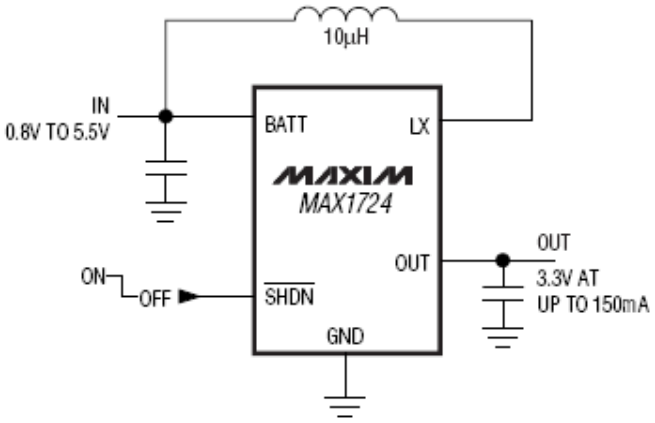
Τα pins του ολοκληρωμένου έχουν την παρακάτω λειτουργία:

Pin1 : BATT, είσοδος μπαταρίας
Pin2 : GND , γη
Pin3 : SHDN, Είσοδος για απενεργοποίηση, όταν έχει υψηλή τάση έχει φυσιολογική λειτουργία, όταν είναι στη γη είναι απενεργοποιημένο
Pin4 : OUT, Τάση εξόδου, επίσης παρέχει bootstrap ισχύ στο ολοκληρωμένο
Pin5 : LX, Εσωτερικός διακόπτης για τα N και P channel Mosfets.

Το απλοποιημένο διάγραμμα λειτουργίας του MAX1724 φαίνεται στο σχήμα 3-4 και το πρότυπο κύκλωμα λειτουργίας στο σχήμα 3-5.



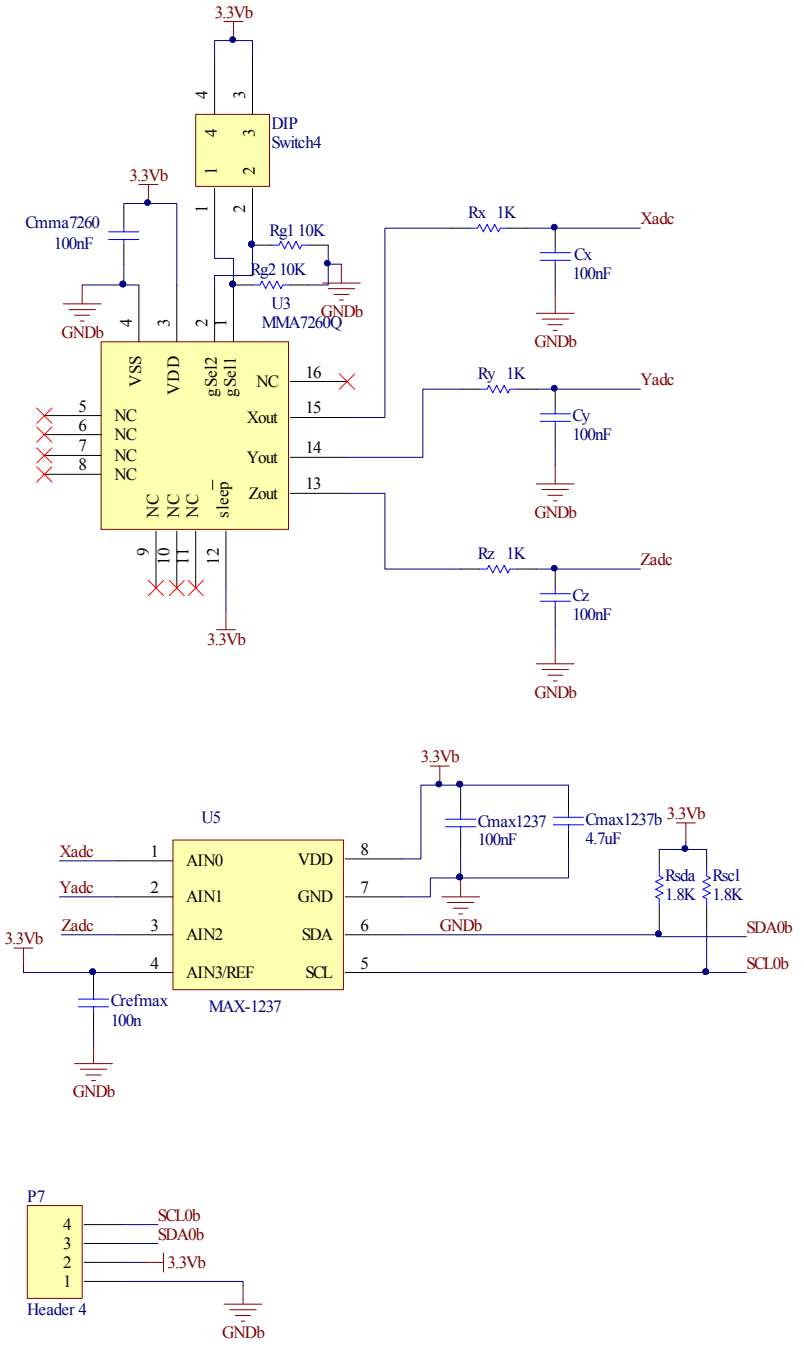
Σχήμα 3-4: Απλοποιημένο διάγραμμα λειτουργίας του MAX1724.



Σχήμα 3-5: Πρότυπο κύκλωμα λειτουργίας του MAX1724.

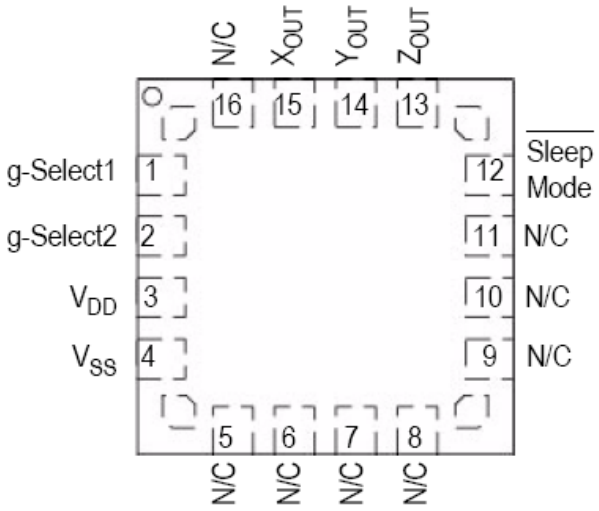
3.2.3 Υποσύστημα επιταχυνσιομέτρου

Το κύκλωμα του επιταχυνσιομέτρου αποτελείται από το επιταχυνσιόμετρο MMA7260 και τον ADC MAX1237. Το MMA7260 δίνει στις εξόδους του μια τάση που είναι ανάλογη με την επιτάχυνση σε κάθε άξονα. Η τάσεις αυτές τροφοδοτούνται στον ADC μέσω βαθυπερατών φίλτρων. Ο ADC κάθε φορά που ο μικροελεγκτής του λεει να διαβάσει τις τιμές των εισόδων του, σαρώνει τις εισόδους και στέλνει τις τιμές τους πίσω στον μικροελεγκτή. Η τροφοδοσία της πλακέτας δίνεται από το υποσύστημα του μικροελεγκτή. Το σχηματικό διάγραμμα του υποσυστήματος του επιταχυνσιομέτρου δίνεται στο σχήμα 3-6.



Σχήμα 3-6: Σχηματικό διάγραμμα υποσυστήματος επιταχυνσιομέτρου.

Το επιταχυνσιόμετρο MMA7260 της Freescale είναι ένα χωρητικό επιταχυνσιόμετρο με εσωτερικό βαθυπερατό φίλτρο ενός πόλου και θερμοκρασιακή αντιστάθμιση. Είναι σε πακέτο QFN 16 ακίδων και η πάνω του όψη φαίνεται στο σχήμα 3-7.



Σχήμα 3-7: Η πάνω όψη του MMA7260.

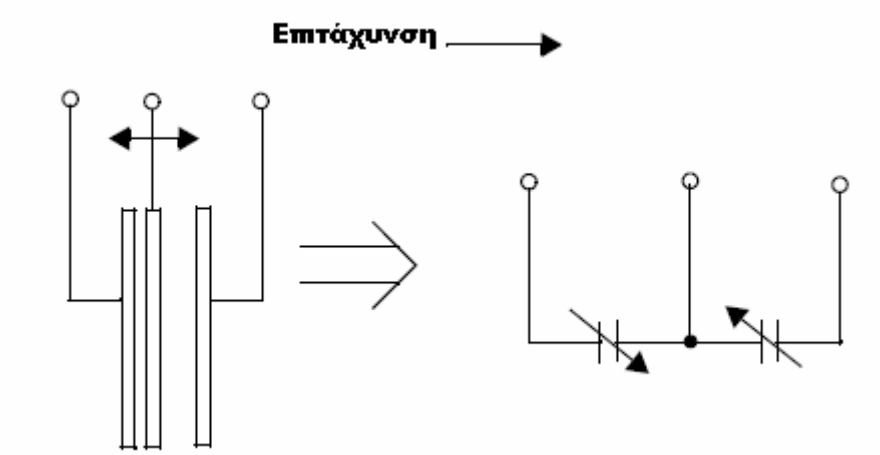
Τα κύρια χαρακτηριστικά του είναι:
-Επιλέξιμη ευαισθησία (1.5g/2g/4g/6g)
-Χαμηλή κατανάλωση ρεύματος (500μA)
-Κατάσταση χαμηλής κατανάλωσης (sleep mode 3μA)
-Χαμηλή τάση λειτουργίας 2.2V-3.6V
-Διαστάσεις 6mm x 6mm x 1.45mm QFN
-Υψηλή ευαισθησία (800 mV/g @ 1.5g)
-Γρήγορη απόκριση εκκίνησης
-Εσωτερική προετοιμασία σήματος με βαθυπερατό φίλτρο
-Γερή σχεδίαση, αντοχή σε δυνατά χτυπήματα
-Οικολογικό πακετάρισμα
-Χαμηλό κόστος

Τα pins του ολοκληρωμένου έχουν την παρακάτω λειτουργία:

Pin 1: g-Select1, Λογική είσοδος για επιλογή ευαισθησίας
Pin 2: g-Select2, Λογική είσοδος για επιλογή ευαισθησίας
Pin 3: VDD, Είσοδος θετικής τροφοδοσίας
Pin 4: VSS, Γη τροφοδοσίας
Pin 5-7:N/C, Χωρίς εσωτερική σύνδεση
Pin 8-11: N/C, Για εργοστασιακό ρύθμισμα
Pin 12: Sleep Mode, Λογική είσοδος για να εισερχεται το ολοκληρωμένο σε κατάσταση χαμηλής κατανάλωσης
Pin 13: Zout, τάση εξόδου για τον Z άξονα
Pin 14: Yout, τάση εξόδου για τον Y άξονα
Pin 15: Xout, τάση εξόδου για τον X άξονα
Pin 16: N/C, Χωρίς εσωτερική σύνδεση

Το MMA7260 αποτελείται από δύο χωρητικά κύτταρα αισθητήρων (g-cell) και ένα κύκλωμα προετοιμασίας σήματος. Το κύτταρο αυτό είναι μια μηχανική δομή από πολυπυρίτιο. Μπορει να μοντελοποιηθεί σαν ένα ζεύγος δεσμών που είναι προσαρτημένα σε μια κινητή κεντρική μάζα η οποία κινείται μεταξύ

των δύο δεσμών. Οι κινητές δέσμες μπορούν να εκτραπούν από την θέση ηρεμίας τους εφαρμόζοντας στο σήμα κάποια επιτάχυνση (σχήμα 3-8).



Σχήμα 3-8: Απλοποιημένο μοντέλο λειτουργίας επιταχυνσιομέτρου.

Όταν οι δέσμες κινηθούν λόγω επιταχύνσεως, η απόσταση τους από τις σταθερές δέσμες της μιας πλευράς θα αυξηθεί ενώ η απόστασή τους από τις σταθερές δέσμες της αντίθετης πλευράς θα μειωθεί κατά τον ίδιο τρόπο. Η αλλαγή της απόστασης είναι ένα μέγεθος επιτάχυνσης.

Οι δέσμες των κυττάρων επιταχύνσεως σχηματίζουν δύο πυκνωτές όπως φαίνεται στο σχήμα 3-8. Καθώς η μεσαία δέσμη κινείται λόγω επιτάχυνσης, η απόσταση μεταξύ των δεσμών αλλάζει και κατά συνέπεια αλλάζει και η χωρητικότητα των δύο πυκνωτών με βάση τη σχέση C=Aε/D, όπου A είναι το εμβαδόν της δέσμης, ε είναι η διηλεκτρική σταθερά και D είναι η απόσταση μεταξύ των δεσμών που σχηματίζουν τον πυκνωτή.

Το ολοκληρωμένο μετράει την χωρητικότητα των κυττάρων επιτάχυνσης και εξάγει την επιτάχυνση από τη διαφορά των δύο πυκνωτών. Επίσης κάνει προετοιμασία σήματος και φιλτράρει το σήμα δίνοντας στην έξοδο τάση η οποία είναι ανάλογη της επιτάχυνσης.

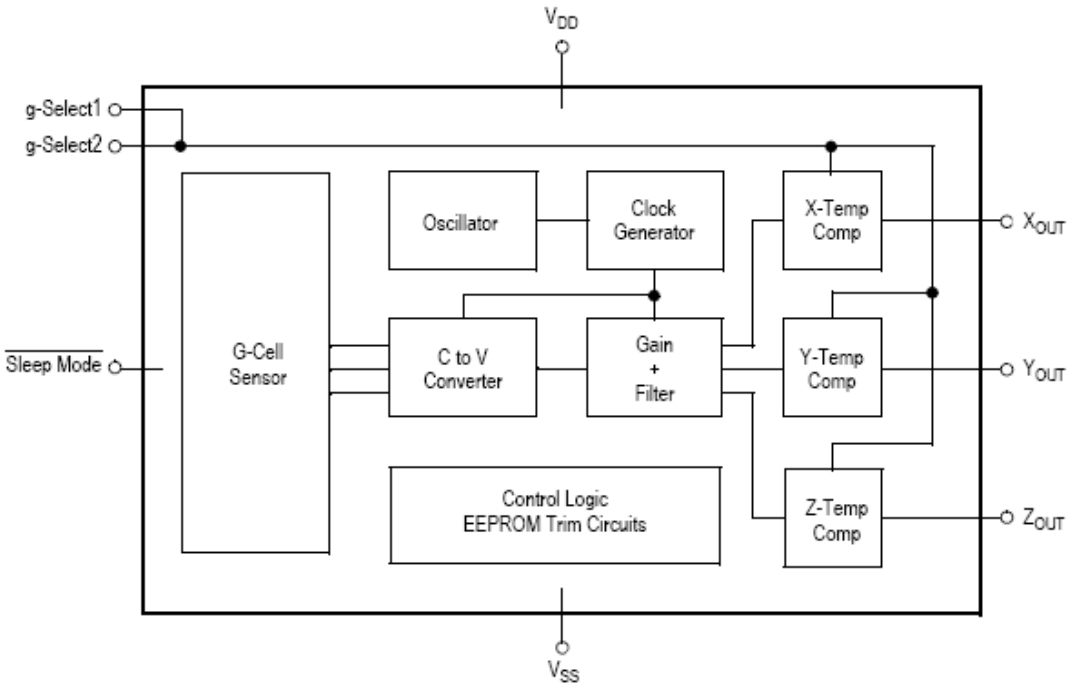
Ανάλογα με την λογική τάση που εφαρμόζεται στις εισόδους g-Select1 και g-Select2 είναι δυνατή η επιλογή ευαισθησίας του αισθητήρα μεταξύ 1.5g, 2g, 4g και 6g (πίνακας 3-a). Αυτό είναι πολύ χρήσιμο καθώς ανάλογα με την εφαρμογή μπορούμε να επιλέξουμε και διαφορετική ευαισθησία. Επιπλέον η ευαισθησία μπορεί να αλλάξει οποιαδήποτε στιγμή κατά την λειτουργία του ολοκληρωμένου. Τα g-Select1 και g-Select2 μπορούν να μείνουν χωρίς σύνδεση για τις εφαρμογές που απαιτούν 1.5g ευαισθησίας καθώς το MMA7260 έχει εσωτερικές pull-down αντιστάσεις για να κρατούν την ευασθησία σε αυτή την τιμή.

g-Select2	g-Select1	g-Range	Sensitivity
0	0	1.5g	800 mV/g
0	1	2g	600 mV/g
1	0	4g	300 mV/g
1	1	6g	200 mV/g

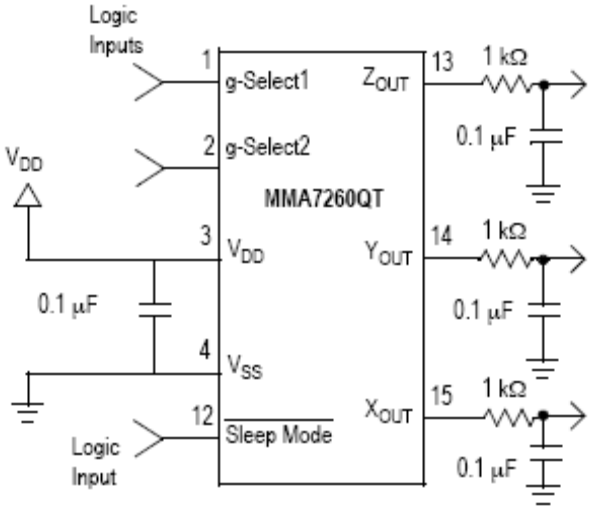
Πίνακας 3-a: Οι ευαισθησίες του MMA7260.

Όταν η είσοδος Sleep Mode είναι ενεργοποιημένη, οι έξοδοι του ολοκληρωμένου απενεργοποιούνται και η κατανάλωση ρεύματος μειώνεται σημαντικά στα 3μA. Αυτό είναι ιδανικό για εφαρμογές που χρησιμοποιούν μπαταρία για την τροφοδοσία τους.

Το απλοποιημένο μπλοκ διάγραμμα λειτουργίας του επιταχυνσιομέτρου φαίνεται στο σχήμα 3-9 ενώ το προτεινόμενο κύκλωμα λειτουργίας του παρουσιάζεται στο σχήμα 3-10.



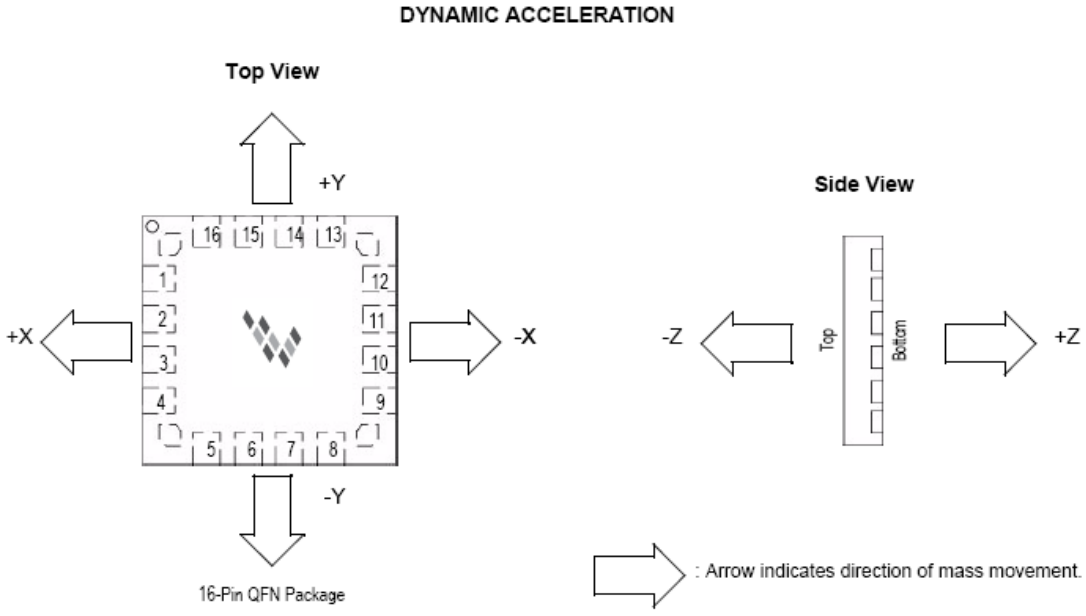
Σχήμα 3-9: Το απλοποιημένο μπλοκ διάγραμμα λειτουργίας του MMA7260.



Σχήμα 3-10: Προτεινόμενο κύκλωμα λειτουργίας του MMA7260.

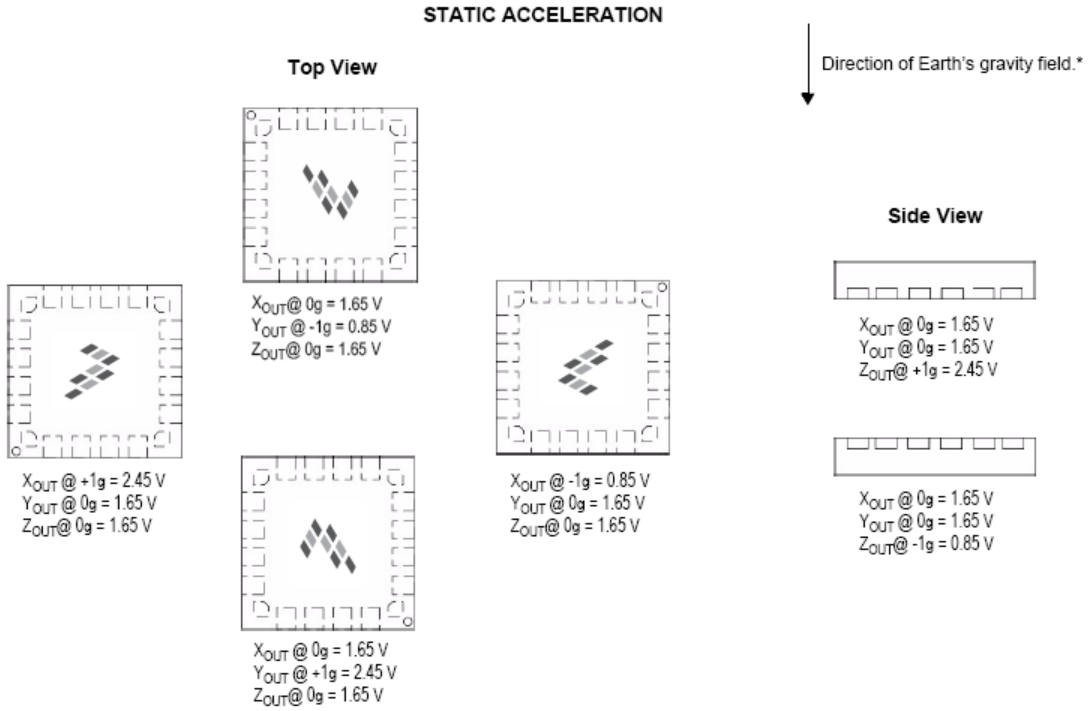
Στο υποσύστημα επιταχυνσιομέτρου που κατασκευάστηκε ακολουθήθηκε το σχήμα 3-10 και οι οδηγίες από το datasheet του ολοκληρωμένου. Χρησιμοποιήθηκε πυκνωτής 100nF κοντά στην τάση τροφοδοσίας του MMA7260, η απόσταση του από τον ADC κρατήθηκε μικρή, χρησιμοποιήθηκε γη (ground plane) κάτω από το επιταχυνσιόμετρο για να μειωθεί ο θόρυβος, συνδέθηκαν RC φίλτρα στις εξόδους για να μειωθεί ο θόρυβος από το εσωτερικό ρολόι των switched capacitor φίλτρων και η συχνότητα δειγματοληψίας του ADC επιλέχτηκε διαφορετική από τα 11kHz, που είναι η εσωτερική συχνότητα δειγματοληψίας του επιταχυνσιομέτρου, για να μην υπάρχουν λάθη λόγω aliasing. Τέλος έχουμε προβλέψει την τοποθέτηση dip διακόπτη για την επιλογή ευαισθησίας του ολοκληρωμένου.

Οι διευθύνσεις των επιταχύνσεων των τριών αξόνων σε σχέση με το ολοκληρωμένο φαίνονται στο σχήμα 3-11.



Σχήμα 3-11: Δυναμική επιτάχυνση του MMA7260.

Στις μετρήσεις μας πρέπει να συνυπολογίζεται και η επιτάχυνση της βαρύτητας της γής (1g). Στο σχήμα 3-12 παρουσιάζονται παραδείγματα για διαφορετικές τοποθετήσεις του ολοκληρωμένου σε σχέση με την επιφάνεια της γης και οι τάσεις εξόδου για του τρεις άξονες.



Σχήμα 3-12: Στατική επιτάχυνση του MMA7260.

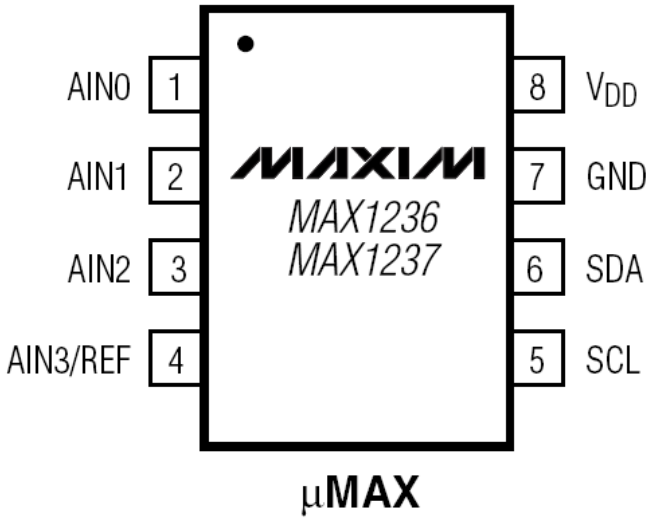
3.2.4 Υποσύστημα αναλογικοψηφιακού μετατροπέα

Το MAX1237 της MAXIM είναι ένας μετατροπέας αναλογικού σήματος σε ψηφιακό (ADC). Έχει 12 bit ακρίβεια, χαμηλή κατανάλωση, 4 κανάλια, τάση αναφοράς και υποστηρίζει επικοινωνία με το πρωτόκολλο I2C. Τα κύρια χαρακτηριστικά του είναι:

-Υψηλής ταχύτητας I2C επικοινωνία
  400kHz Fast mode
  1.7MHz High-Speed Mode
-Μονή τροφοδοσία 2.7V ως 3.6V
-Εσωτερική τάση αναφοράς: 2.048V
-Εξωτερική τάση αναφοράς : 1V μέχρι VDD
-Εσωτερικό ρολόι
-4 μονά κανάλια ή 2 διαφορικά κανάλια για δειγματοληψία
-Εσωτερικός καταχωρητής FIFO με Channel-Scan mode
-Χαμηλή κατανάλωση ανάλογα με την συχνότητα δειγματοληψίας
  670μA στα 94.4ksps
  230μA στα 40ksps
  60μA στα 10ksps
  6μA στα 1ksps

-0.5μA όταν είναι στην κατάσταση χαμηλής κατανάλωσης(Power-Down Mode)
-Μικρό πακετάρισμα 8-pin μMAX

Η πάνω όψη του MAX1237 παρουσιάζεται στο σχήμα 3-13.



Σχήμα 3-13: Η πάνω όψη του MAX1237.

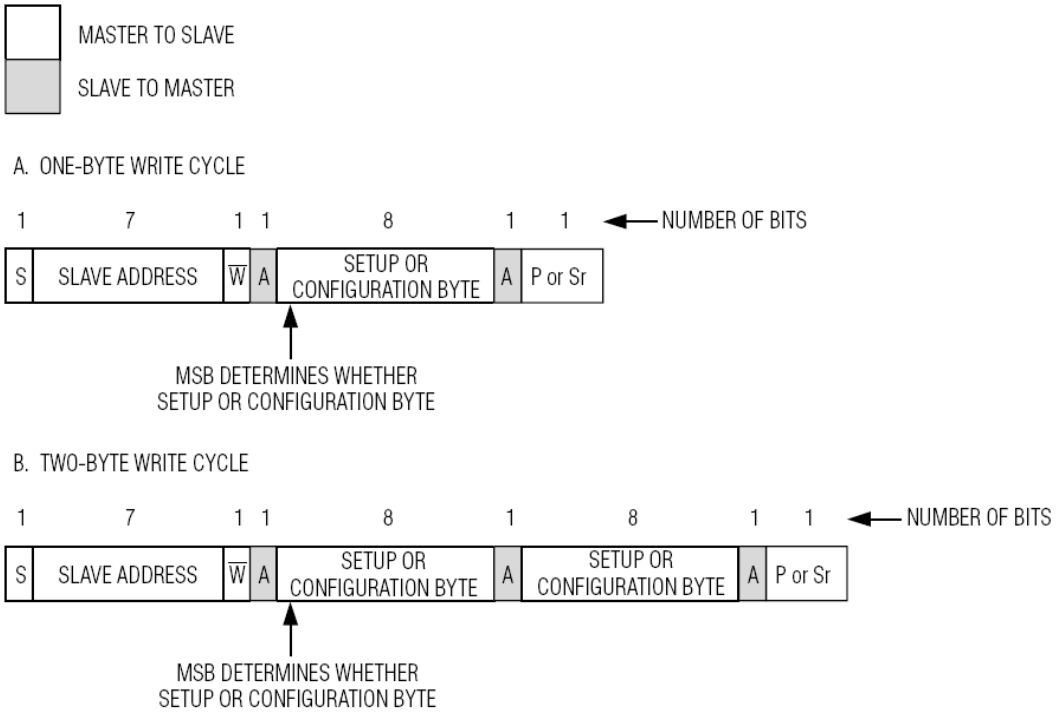
Τα πινς του ολοκληρωμένου εχουν την παρακάτω λειτουργία:

Pin 1: AIN0, Αναλογική είσοδος 0
Pin 2: AIN1, Αναλογική είσοδος 1
Pin 3: AIN2, Αναλογική είσοδος 2
Pin 4: AIN3/REF, Αναλογική είσοδος 3, είσοδος τάσης αναφοράς ή έξοδος
Pin 5: SCL, Είσοδος ρολογιού (I2C)
Pin 6: SDA, Είσοδος/Έξοδος δεδομένων (I2C)
Pin 7: GND, Γη
Pin 8: VDD, Θετική τάση τροφοδοσίας

Το MAX1237 χρησιμοποιεί successive-approximation τεχνική και διαφορικό track/hold κύκλωμα για την μετατροπή του αναλογικού σήματος σε ψηφιακό των 12 bit. Η επικοινωνία με τον ADC γίνεται μέσω διασύνδεσης δύο καλωδίων (I2C) και η μεταφορά δεδομένων φτάνει τα 1.7MHz.

Η αρχιτεκτονική του MAX1237 βασίζεται σε έναν πολυπλέκτη με αναλογικές εισόδους, έναν πυκνωτή track-and-hold (T/H), T/H διακόπτες, έναν συγκριτή και έναν μετατροπέας ψηφιακού σήματος σε αναλογικό (DAC). Επίσης περιέχει εσωτερικές διόδους στην είσοδο των αναλογικών σημάτων ώστε να ψαλιδίζουν το σήμα μεταξύ 0V και VDD. Με αυτό τον τρόπο επιτρέπεται στο αναλογικό σήμα να πάρει τιμές 0V-0,3V με VDD+0.3V χωρίς να υπάρχει κίνδυνος για το ολοκληρωμένο. Για μετρήσεις ακριβείας το αναλογικό σήμα δεν πρέπει να παίρνει τιμές μεγαλύτερες των 50mV πάνω από VDD ή κάτω από τα 0V της γης.

Ο έλεγχος της λειτουργίας του MAX1237 γίνεται μέσω του I2C. Ο ADC λειτουργεί σαν slave και ο μικροελεγκτής σαν master. Για την λειτουργία του μετατροπέα ο μικροελεγκτής πρέπει να στείλει ένα setup byte ή configuration byte στο MAX1237 με όποια σειρά θέλει, όπως φαίνεται και στο σχήμα 3-14.



Σχήμα 3-14: Τρόποι εγγραφής στο MAX1237.

BIT 7 (MSB)	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 (LSB)
REG	SEL2	SEL1	SEL0	CLK	BIP/$\overline{\text{UNI}}$	RST	X

Σχήμα 3-15: Δομή Setup Byte του MAX1237.

Το Setup Byte έχει την δομή που φαίνεται στο σχήμα 3-15.
Το bit7 καθορίζει αν στέλνουμε setup ή configuration byte. Για setup byte πρεπει να έχει την τιμή 1.
Τα bit6,bit5,bit4 καθορίζουν την τάση αναφοράς και την λειτουργία του AIN/REF pin σύμφωνα με τον πίνακα 4-b.
Το bit3 καθορίζει αν το ρολόι θα είναι εσωτερικό(=0) ή εξωτερικό(=1).
Το bit2 καθορίζει αν ο ADC θα λειτουργεί σε unipolar(=0) ή bipolar(=1) mode.
Όταν το bit1 έχει την τιμή 0 γίνεται reset στον configuration καταχωρητή. Όταν είναι 1 δεν γίνεται τίποτα.
Το bit0 δεν χρησιμοποιείται.

SEL2	SEL1	SEL0	REFERENCE VOLTAGE	AIN_/REF	INTERNAL REFERENCE STATE
0	0	X	V_{DD}	Analog Input	Always Off
0	1	X	External Reference	Reference Input	Always Off
1	0	0	Internal Reference	Analog Input	Always Off
1	0	1	Internal Reference	Analog Input	Always On
1	1	0	Internal Reference	Reference Output	Always Off
1	1	1	Internal Reference	Reference Output	Always On

Πίνακας 3-b: Καθορισμός τάσης αναφοράς και AIN/REF pin.

BIT 7 (MSB)	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 (LSB)
REG	SCAN1	SCAN0	CS3	CS2	CS1	CS0	SGL/$\overline{DIF}$

Σχήμα 3-16: Δομή configuration byte.

Το configuration byte έχει την δομή που φαίνεται στο σχήμα 3-16.
Το Bit7 καθορίζει αν το byte που στέλνουμε είναι configuration(=0) ή setup(=1) byte.
Το Bit6 και Bit5 καθορίζουν τον τρόπο που διαβάζονται τα αναλογικά κανάλια όπως φαίνεται στον πίνακα 3-c.
Τα Bit 4,3,2,1 καθορίζουν ποια κανάλια να διαβαστούν από το MAX1237 όπως φαίνεται στον πίνακα 3-d.
Το Bit0 καθορίζει αν τα κανάλια θα είναι μονά (single-ended =1) ή διαφορικά (differential =0). Πίνακας 3-d.

SCAN 1	SCAN 0	Τρόπος ανάγνωσης καναλιών
0	0	Διαβάζει από το AIN0 μέχρι την είσοδο που καθορίζουν τα CS3-CS0.Όταν το AIN/REF πιν είναι τάση αναφοράς,η ανάγνωση σταματάει στο AIN3.
0	1	Μετατρέπει την είσοδο που έχει επιλεγεί από τα CS3-CS0 οκτώ φορές.
1	0	Διαβάζει από το AIN2 μέχρι την είσοδο που καθορίζουν τα CS1-CS0.Όταν τα CS1-CS0 έχουν επιλεγεί για AIN0-AIN2,το διάβασμα σταματάει στο AIN2.Όταν το AIN/REF πιν είναι καθορισμένο σαν τάση αναφοράς το διάβασμα σταματάει στο AIN3.
1	1	Μετατρέπει το κανάλι που έχει επιλεγεί από τα CS3-CS0.

Πίνακας 3-c: Καθορισμός ανάγνωσης καναλιών.

CS3[1]	CS2[1]	CS1	CS0	AIN0	AIN1	AIN2	AIN3[2]
0	0	0	0	+			
0	0	0	1		+		
0	0	1	0			+	
0	0	1	1				+

Όταν SGL/DIF=1

CS3[1]	CS2[1]	CS1	CS0	AIN0	AIN1	AIN2	AIN3[2]
0	0	0	0	+	-		
0	0	0	1	-	+		
0	0	1	0			+	-
0	0	1	1			-	+

Όταν SGL/DIF=0

Πίνακας 3-d: Επιλογή καναλιών.

Μονή και διαφορική είσοδος (Single-ended/differential input).
Το SGL/DIF bit του configuration byte καθορίζει αν οι αναλογικές είσοδοι θα διαβαστούν σαν μονές ή διαφορικές. Όταν SGL/DIF=1 έχουμε μονές εισόδους και το αποτέλεσμα της ψηφιακής μετατροπής είναι μεταξύ της αναλογικής εισόδου που καθορίζουν τα CS[3:0] bit και της γης. Όταν SGL/DIF=0 έχουμε διαφορικές εισόδους και το αποτέλεσμα της ψηφιακής μετατροπής είναι η διαφορά μεταξύ των '+' και '-' των αναλογικών εισόδων που καθορίζονται από τα CS[3:0] bit. (πίνακας 3-d).

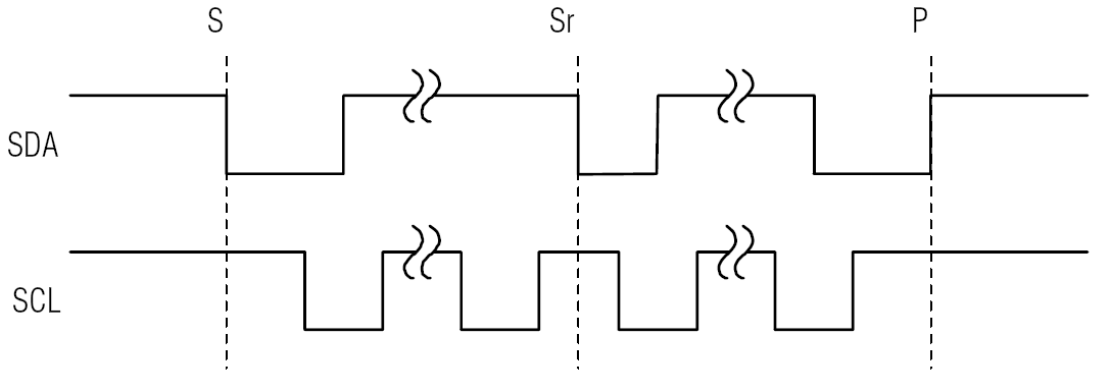
Μονοπολική και διπολική λειτουργία (Unipolar/Bipolar).
Όταν έχουμε επιλέξει διαφορική λειτουργία, το BIP/UNI bit του Setup byte καθορίζει αν θα έχουμε μονοπολική ή διπολική λειτουργία. Η μονοπολική λειτουργία καθορίζει το πεδίο τιμών της διαφορικής εισόδου από τα 0V μέχρι το VREF της τάσης αναφοράς (σχήμα 3-23). Μια αρνητική διαφορική αναλογική είσοδος στη μονοπολική λειτουργία δίνει μηδενικό ψηφιακό αποτέλεσμα. Η διπολική λειτουργία καθορίζει το πεδίο τιμών της διαφορικής εισόδου από τα –VREF/2 μέχρι τα +VREF/2 (σχήμα 3-24). Το ψηφιακό αποτέλεσμα είναι δυαδικό στη μονοπολική λειτουργια και συμπλήρωμα ως προς δύο στην διπολική λειτουργία. Στην μονή λειτουργία των εισόδων, το MAX1237 λειτουργεί πάντα με μονοπολικό τρόπο ανεξαρτήτως της τιμής του BIP/UNI. Οι αναλογικές είσοδοι έχουν αναφορά τη γη και πεδίο τιμών εισόδου είναι 0V μέχρι VREF.

Μεταφορά δεδομένων μέσω πρωτοκόλλου δύο καλωδίων (I2C).
Το MAX1237 χρησιμοποιεί το πρωτόκολλο I2C για την μεταφορά δεδομένων. Το SDA χρησιμοποιείται για την μεταφορά πληροφορίας από τον master προς τον slave αλλά και το αντίστροφο με ταχύτητες που φτάνουν τα 1.7MHz. Το

SCL είναι το ρολόι της μεταφοράς που δημιουργεί ο master. Τα SDA και SCL πρέπει να έχουν υψηλή ονομαστική τάση. Αυτό γίνεται μέσω pull-up αντιστάσεων (750Ω ή μεγαλύτερες).

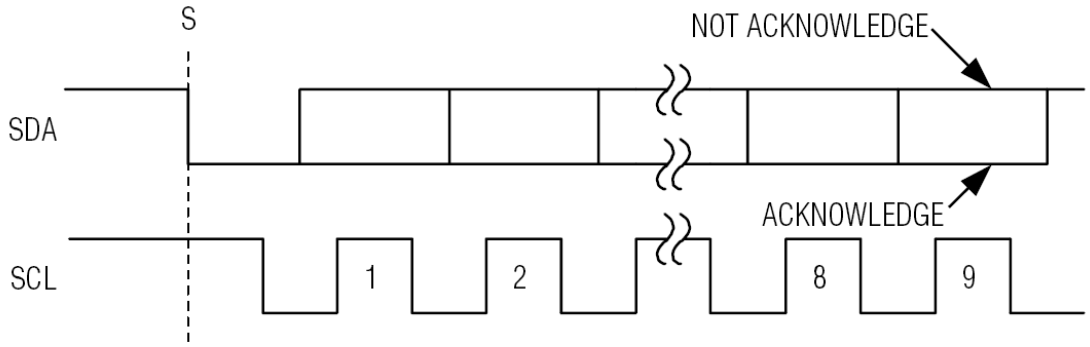
Ένα bit δεδομένων μεταφέρεται κατά τη διάρκεια ένος SCL κύκλου ρολογιού. Χρειάζονται 18 κύκλοι ρολογιού το ελάχιστο για μεταφορά δεδομένων πρός ή από το MAX1237.Η SDA γραμμή πρέπει να παραμένει σταθερή κατά τη διάρκει που το SCL έχει υψηλή τάση. Αλλαγές στην SDA γραμμή όταν το SCL είναι σταθερό θεωρούνται σήματα ελέγχου. Η SDA και η SCL γραμμή είναι σε υψηλή τάση όταν ο διάδρομος δεδομένων είναι ελεύθερος.

Ο master όταν θέλει να ξεκινήσει μια αποστολή, δημιουργεί μια START κατάσταση (S) όταν αλλάζει την SDA γραμμή από υψηλή τάση σε χαμηλή όταν η SCL γραμμή είναι σε υψηλή τάση. Όταν θέλει να τερματίσει την επικοινωνία στέλνει μια STOP κατάσταση (P), δηλαδή αλλάζει την SDA γραμμή από χαμηλή τάση σε υψηλή όταν η SCL γραμμή είναι σε υψηλή τάση. Μια επαναλαμβανόμενη START κατάσταση (Sr) μπορεί να χρησιμοποιηθεί αντί της STOP κατάστασης όταν ο master θέλει να να αφήσει τον διάδρομο δεδομένο ενεργό.



Σχήμα 3-17: START και STOP καταστάσεις.

Οι μεταφορές δεδομένων αναγνωρίζονται με το bit αναγνώρισης (A) ή δεν αναγνωρίζονται με το bit μη-αναγνώρισης (A'). Και ο master και ο slave μπορούν να δημιουργήσουν αυτά τα bit. Για να δημιουργηθεί το bit αναγνώρισης, ο αποδεκτης δεδομένων πρέπει να κρατήσει την SDA γραμμή σε χαμηλή τάση πριν την θετική ακμή του 9^{ου} παλμού του SCL και να την κρατήσει χαμηλή κατά την διάρκεια που ο SCL παλμός είναι σε υψηλή τάση. Για να δημιουργηθεί bit μη-αναγνώρισης, ο δέκτης αφήνει την SDA γραμμή να πάει σε υψηλή τάση πριν την θετική ακμή του 9^{ου} παλμού του SCL και να μείνει σε υψηλή κατάσταση κατά τη διάρκεια που ο SCL παλμός είναι σε υψηλή τάση. Παρακολουθόντας τα bit αναγνώρισης μπορούμε να καταλάβουμε πότε έγινε επιτυχημένη μεταφορά δεδομένων. Αν προέκυψε κάποιο λάθος κατά τη μεταφορά, ο master πρέπει να δοκιμάσει να επικοινωνήσει ξανά με τον slave κάποια άλλη στιγμή. (Σχήμα 3-18).



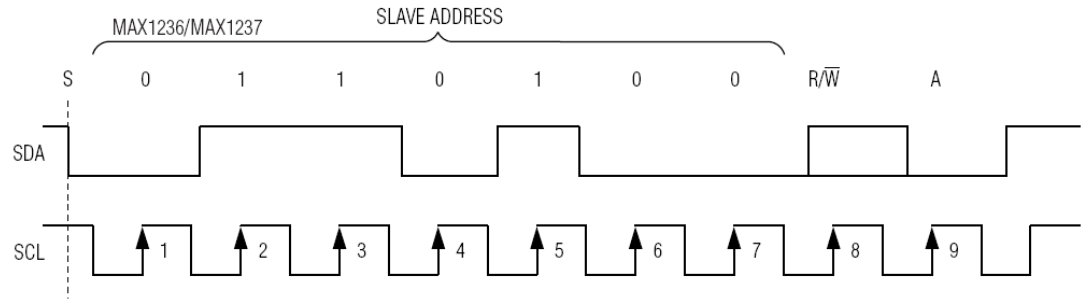
Σχήμα 3-18: Bit αναγνώρισης και μη-αναγνώρισης.

Ο master του διαδρόμου δεδομένων ξεκινάει την επικοινωνία με τον slave δημιουργόντας μια START κατάσταση και μετά στέλνοντας την διεύθυνση του slave στον οποίο απευθύνεται. Όταν το MAX1237 βρίσκεται σε αδράνεια περιμένει συνεχώς μια START κατάσταση και την διεύθυνσή του. Όταν αυτό πραγματοποιηθεί το MAX1237 είναι έτοιμο να στείλει και να λάβει δεδομένα. Η διεύθυνση του MAX1237 εξαρτάται από τον τύπο του ολοκληρωμένου που χρησιμοποιούμε (πίνακας 3-e).

Ολοκληρωμένο	I2C διεύθυνση
MAX1237EUA	110100
MAX1237KEUA	110000
MAX1237LEUA	110010
MAX1237MEUA	110110

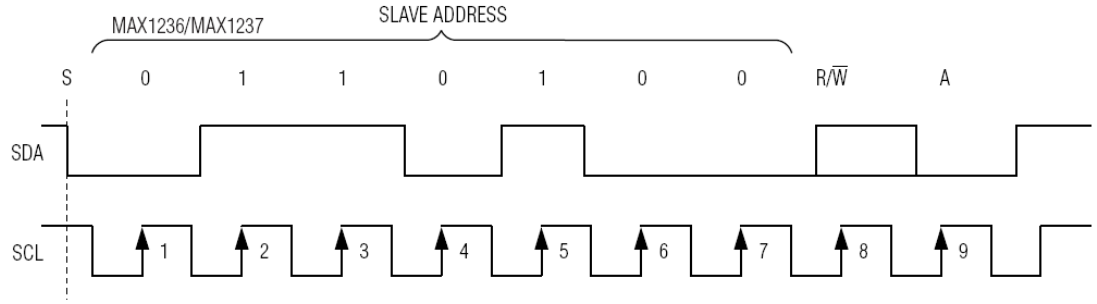
Πίνακας 3-e: Τύποι ολοκληρωμένων MAX1237 και οι I2C διευθύνσεις τους.

Το μικρότερης σημασίας bit (LSB) του byte διεύθυνσης (R/W) καθορίζει αν ο master διαβάζει από τον slave (R/W =1) ή γράφει στον slave (R/W =0). Αφού το MAX1237 δεχτεί την διεύθυνσή του, δημιουργεί ένα bit αναγνώρισης (A). Όλα τα παραπάνω φαίνονται στο σχήμα 3-19.



Σχήμα 3-19: Byte διεύθυνσης του MAX1237.

Κατά την εκκίνηση του MAX1237 ο χρονισμός του διαδρόμου δεδομένων είναι σε γρήγορη κατάσταση (Fast mode-F/S), ο οποίος επιτρέπει ρυθμό μετατροπής μέχρι 22.2ksps. Το MAX1237 πρέπει να λειτουργήσει σε υψηλής ταχύτητας κατάσταση (High Speed mode-H/S) ώστε να μπορεί να φτάσει ρυθμούς μετατροπής μέχρι 94.4ksps. Για να αρχίσει να λειτουργεί το MAX1237 σε H/S mode πρέπει ο master να στείλει το byte 00001XXX (X=0 ή 1). Αφού το MAX1237 λάβει αυτόν τον κωδικό δημιουργεί ένα bit μη-αναγνώρισης και εισέρχεται σε H/S κατάσταση. Μετά από αυτό, ο master πρέπει να δημιουργήσει μια επαναλαμβανόμενη START κατάσταση (Sr) και μετά να στείλει την διεύθυνση του slave ώστε να αρχίσει επικοινωνία σε H/S κατάσταση. An ο master δημιουργήσει STOP κατάσταση, το MAX1237 επιστρέφει σε F/S mode. Τα παραπάνω φαίνονται στο σχήμα 3-20.



Σχήμα 3-20: Μετατροπή από F/S κατάσταση σε H/S κατάσταση.

Ένας κύκλος εγγραφής ξεκινάει με τον master να δημιουργεί μια START κατάσταση. Αμέσως μετά στέλνει την διεύθυνση του MAX1237 και το R/W bit (=0). Αν το byte διεύθυνσης λήφθηκε με επιτυχία, το MAX1237 δημιουργεί ένα bit αναγνώρισης (A). Τότε ο master γράφει στον slave. Το MAX1237 αναγνωρίζει το byte που πήρε σαν setup byte αν το πιο σημαντικό bit (MSB) είναι 1 και σαν configuration byte αν το MSB είναι 0. Ο master μπορεί να γράψει ένα ή δύο byte, με όποια σειρά θέλει. Όταν ο slave λάβει το byte με επιτυχία δημιουργεί ένα bit αναγνώρισης. Τότε ο master σταματάει τον κύκλο εγγραφής δημιουργώντας STOP κατάσταση (επιστρέφοντας σε F/S mode) ή επαναλαμβανόμενης START κατάστασης (Σχήμα 3-14).

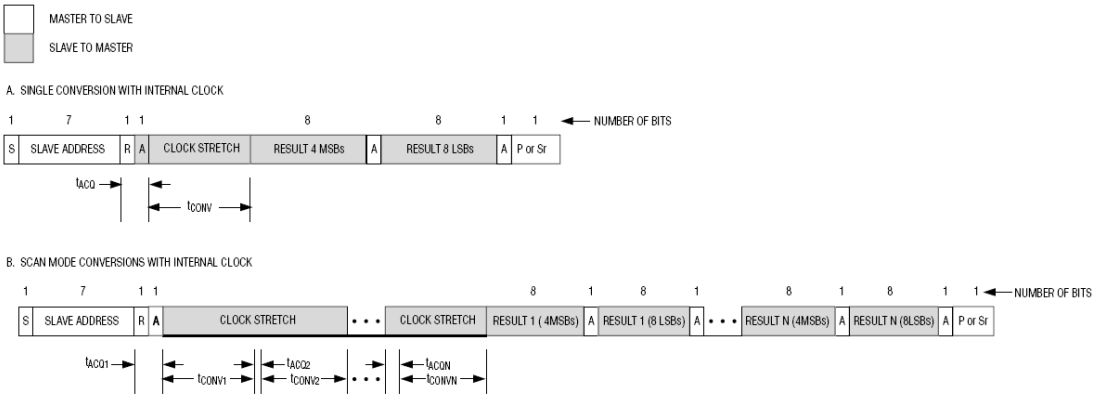
Ο κύκλος ανάγνωσης γίνεται για να λάβει ο master τα αποτελέσματα της δειγματοληψίας. Για να γίνει αυτό, ο master δημιουργεί μια START κατάσταση και στη συνέχεια στέλνει την διεύθυνση του MAX1237 με το R/W bit =1. Αν το byte διεύθυνσης λήφθηκε με επιτυχία, το MAX1237 δημιουργεί ένα bit αναγνώρισης. Μετά ο master διαβάζει τα δεδομένα. Το αποτέλεσμα μεταδίδεται σε δύο byte. Πρώτα τα 4 bit του πρώτου byte είναι 1 και μετά στέλνεται το αποτέλεσμα,πρώτα το MSB και τέλος το LSB. Όταν ο master λάβει τα δεδομένα που θέλει μπορεί να στείλει bit αναγνώρισης αν επιθυμεί να συνεχίσει να διαβάζει ή bit μη-αναγνώρισης αν δεν θέλει. Αν το MAX1237 λάβει bit μη-αναγνώρισης, ελευθερώνει την SDA γραμμή ώστε ο master να δημιουργήσει STOP ή επαναλαμβανόμενη START κατάσταση.

Η επιλογή ρολογιού καθορίζει το ρολόι μετατροπής, την δειγματοληψία και τον χρόνο μετατροπής. Από το CLK bit του Setup byte μπορούμε να επιλέξουμε εσωτερικό ρολόι (CLK=0) ή εξωτερικό (CLK=1). Κατά την εκκίνηση του

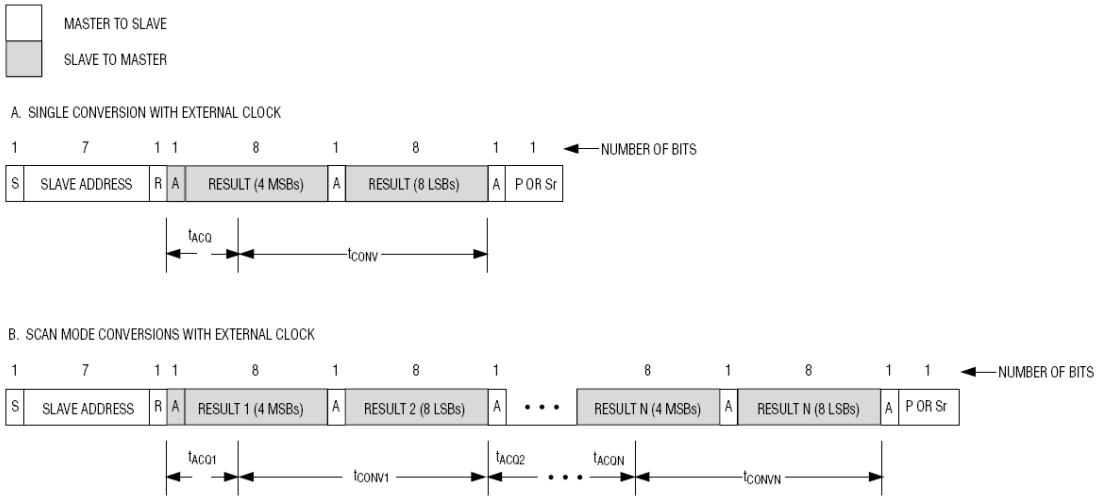
MAX1237 η ρύθμιση είναι σε εσωτερικό ρολόι. Όταν επιλέξουμε την λειτουργία με εξωτερικό ρολόι, το MAX1237 χρησιμοποιεί την SCL γραμμή για ρολόι μετατροπής.

Η επιλογή αν θα χρησιμοποιήσουμε εσωτερικό ή εξωτερικό ρολόι βασίζεται στις προδιαγραφές του συστήματος. Για μετατροπές δεδομένων από 40ksps μέχρι 94.4ksps πρέπει να χρησιμοποιήσουμε εξωτερικό ρολόι. Κάτω όμως από τα 40ksps προτείνεται η λειτουργία με εσωτερικό ρολόι για μικρότερη κατανάλωση ισχύος. Επιπλέον αν η περίοδος του SCL είναι μεγαλύτερη από 60μs πρέπει να χρησιμοποιήσουμε εσωτερικό ρολόι.

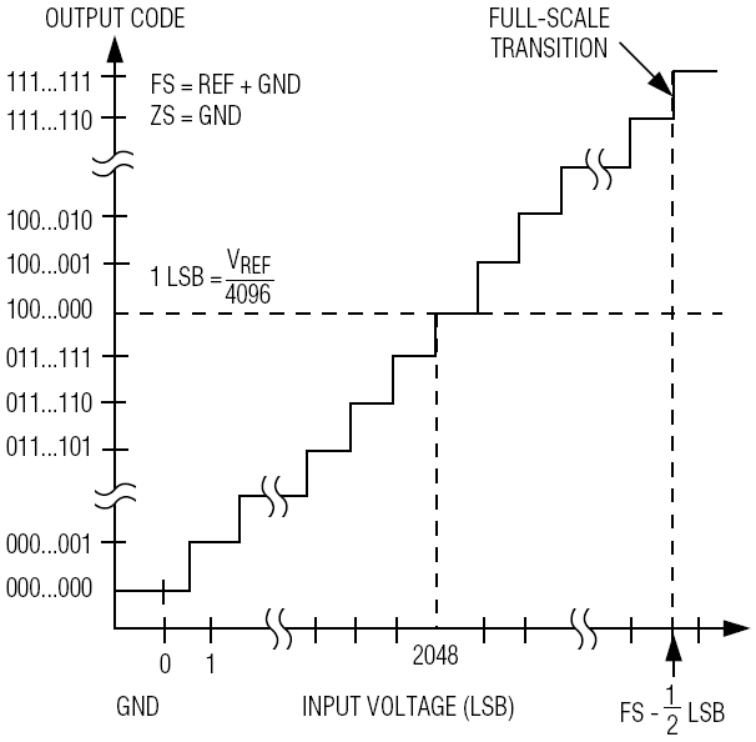
Τα SCAN0 και SCAN1 του configuration byte καθορίζουν τον τρόπο ανάγνωσης των αναλογικών εισόδων. Αν το AIN/REF pin έχει καθοριστεί σαν τάση αναφοράς (SEL=1), τότε εξαιρείται από την ανάγνωση. Τα αποτελέσματα της δειγματοληψίας γράφονται στην μνήμη με τον ίδια σειρα που διαβάζονται. Κάθε αποτέλεσμα χρειάζεται δύο bytes για την αποθήκευσή του. Το σχήμα 3-21 δείχνει τον τρόπο ανάγνωσης όταν χρησιμοποιείται εσωτερικό ρολόι και το σχήμα 3-22 όταν χρησιμοποιείται εξωτερικό.



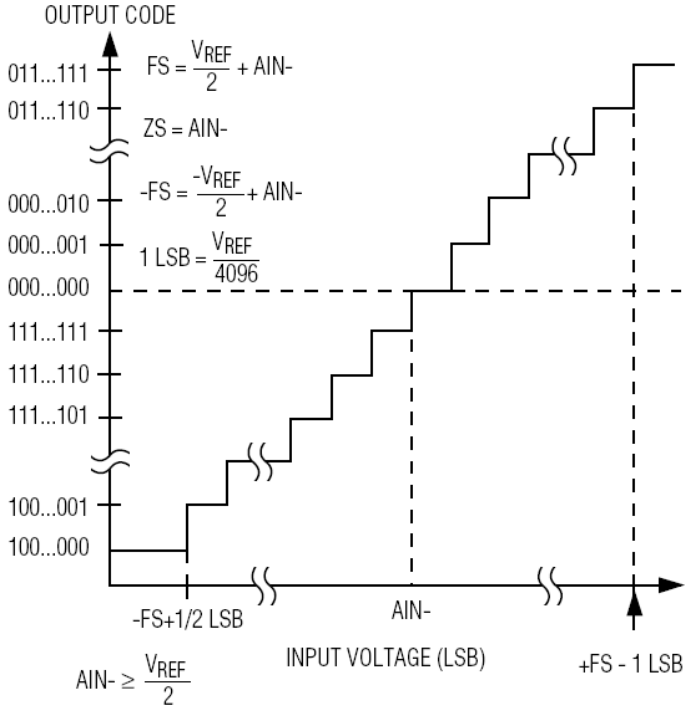
Σχήμα 3-21: Ανάγνωση με εσωτερικό ρολόι.



Σχήμα 3-22: Ανάγνωση με εξωτερικό ρολόι.

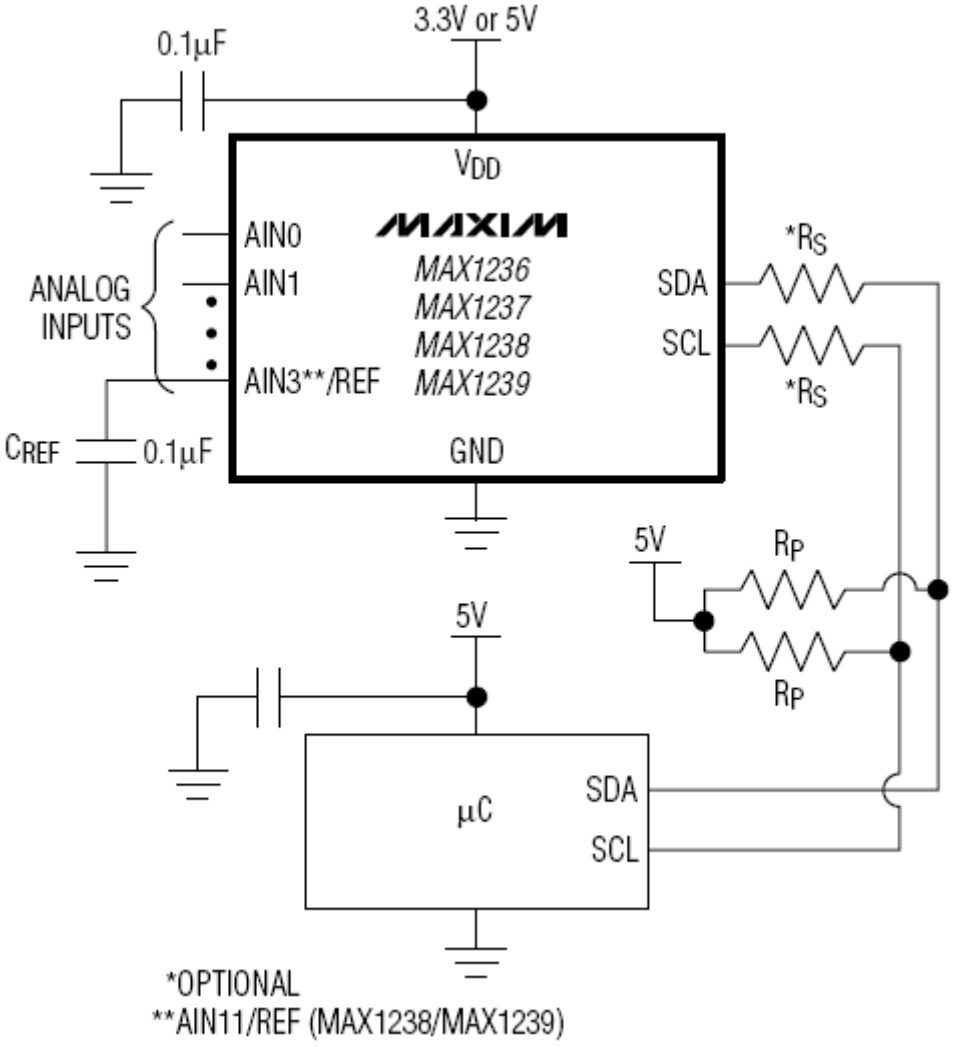


Σχήμα 3-23: Διάγραμμα μετατροπής μονοπολικής λειτουργίας.



Σχήμα 3-24: Διάγραμμα μετατροπής διπολικής λειτουργίας.

Το τυπικό σχηματικό διάγραμμα λειτουργίας για το MAX1237 φαίνεται στο σχήμα 3-25. Στην πλακέτα που κατασκευάσαμε χρησιμοποιήσαμε pull-up αντιστάσεις 1kΩ. Επίσης βάλαμε πυκνωτές 100nF και 4.7μF στο VDD της τροφοδοσίας και 100nF στο AIN/REF πιν. Φροντίσαμε οι πυκνωτές να είναι κοντά στα πινς του ολοκληρωμένου και επιπλέον τοποθετήσαμε γή κάτω από το MAX1237 για να ελαχιστοποιηθεί ο θόρυβος. Τα ψηφιακά σήματα τοποθετήθηκαν πιο μακρυά από τα αναλογικά για να μην υπάρχουν παρεμβολές.



Σχήμα 3-25: Προτεινόμενος τρόπος σύνδεσης του MAX1237 με τον μικροελεγκτή.

ΚΕΦΑΛΑΙΟ 4

Μικροελεγκτής LPC2148

4.1 Γενικά

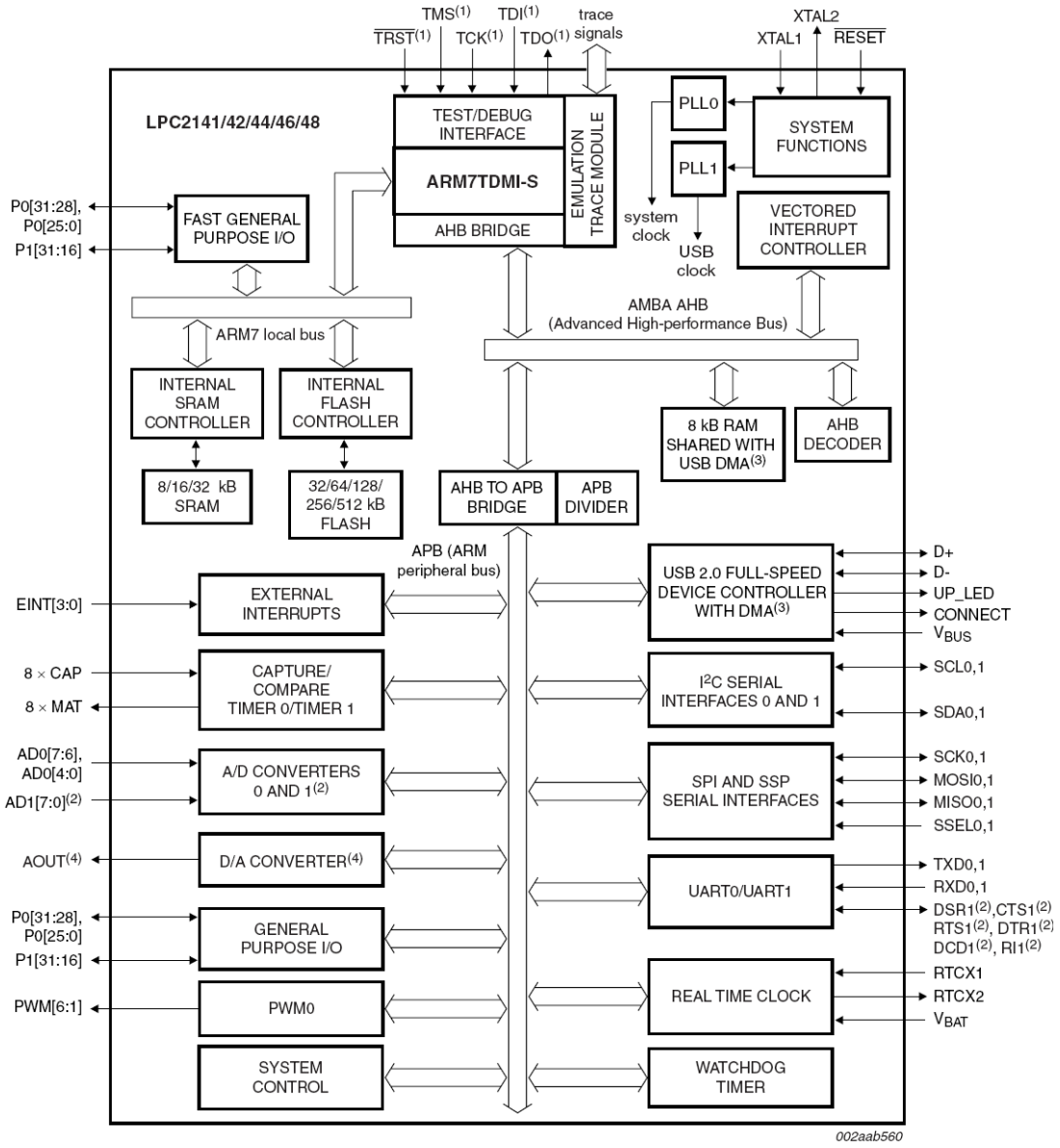
Ο μικροελεγκτής LPC2148 της NXP(Philips) βασίζεται στην 32/16 bit ARM7TDMI-S CPU με real-time emulation και embedded trace support, που συνδυάζει τον μικροελεγκτή με ενσωματωμένη υψηλής ταχύητας μνήμη flash 512kB. Έχει ευρεία διασύνδεση μνήμης των 128-bit και μοναδική αρχιτεκτονική επιτάχυνσης που επιτρέπει εκτέλεση εντολών 32-bit στο μέγιστο δυνατό ρυθμό ρολογιού. Για εφαρμογές που είναι σημαντική η εξοικονόμηση χώρου κώδικα, υπάρχει ο εναλλακτικός τρόπος λειτουργίας σε 16-bit Thumb mode που μειώνει τον κώδικα περισσότερο από 30% με ελάχιστή μείωση στην απόδοση.

Επειδή έχει μικρό μέγεθος και χαμηλή κατανάλωση είναι ιδανικός για εφαρμογές που η σμίκρυνση είναι πολύ σημαντική. Έχει πολλά περιφερειακά για επικοινωνία όπως USB 2.0 πλήρης ταχύτητας, δύο UART, SPI, SSP και I2C. Επιπλέον έχει εσωτερική SRAM 32kB (αυτό δίνει την δυνατότητα μεγάλης προσωρινής αποθήκευσης δεδομένων), διάφορους 32-bit μετρητές, μονούς ή διπλούς 10-bit ADC's, 10-bit DAC, κανάλια PWM και 45 γρήγορες γραμμές εισόδου-εξόδου (GPIO) με μέχρι 9 διακοπές (interrupts) σε στάθμη τάσης και ακμής. Το block διάγραμμα του LPC2148 φαίνεται στο σχήμα 4-1.

Τα κύρια χαρακτηριστικά του είναι:

- 16/32-bit ARM7TDMI-S μικροελεγκτής σε μικρή LQFP64 συσκευασία.
- 32kB εσωτερικής static RAM και 512kB εσωτερικής FLASH μνήμης.
- Ταχύτητα λειτουργίας μέχρι 60MHz χάρη στον 128-bit επιταχυντή.
- Προγραμματισμός πάνω στο κύκλωμα/εφαρμογή ISP/IAP (In-System/In-Application Programming) μέσω boot-loader πρόγραμμα που περιέχει ο μικροελεγκτής.
- Υπάρχει η δυνατότητα αποσφαλμάτωσης σε πραγματικό χρόνο μεσω τής EmbeddedICE RT και Embedded Trace διασύνδεσης.
- Πλήρης ταχύτητας USB 2.0 ελεγκτής με 2kB RAM και επιπλέον 8kB της SRAM του μC χρησιμοποιόντας το DMA.
- Δύο 10-bit αναλογικοψηφιακοί μετατροπείς (ADC) που παρέχουν συνολικά 14 αναλογικές εισόδους, με μικρούς χρόνους μετατροπής 2.44μs ανά κανάλι.
- Ένας μετατροπέας ψηφιακού σήματος σε αναλογικό (DAC) 10-bit που δίνει την δυνατότητα εξόδου μεταβλητής τάσης.

- Δυο μετρητές 32-bit με 4 κανάλια σύλληψης (capture) εξωτερικών γεγονότων και 4 κανάλια σύγκρισης (compare) ο καθένας, μονάδα PWM (pulse-width modulation) 6 εξόδων και watchdog.
- Χαμηλής κατανάλωσης ρολόι πραγματικού χρόνου (RTC) με ανεξάρτητη τροφοδοσία και 32kHz ρολόι εισόδου.
- Πολλαπλές σειριακές διασυνδέσεις όπως 2 UART, 2 γρήγορους I^2C ελεγκτές (400kbit/s), SPI, SSP με μεταβλητό μήκος δεδομένων.
- Ελεγκτής διανυσμάτων διακοπών (vectored interrupt controller) με διαμόρφωση των προτεραιοτήτων και των διευθύνσεων των διακοπών.
- Μέχρι 45 γενικού σκοπού εισόδους-εξόδους με αντοχή έως 5V.
- Μέχρι 9 εξωτερικές διακοπές, ευαίσθητες σε επίπεδο τάσής ή ακμής.
- 60MHz μέγιστη συχνότητα λειτουργίας CPU διαθέσιμη από ενσωματωμένο PLL με χρόνο ηρεμίας 100μs.
- Ενσωματωμένο ολοκληρωμένο ταλαντωτή που λειτουργεί με εξωτερικό κρύσταλλο από 1MHz μέχρι 30MHz και με εξωτερικό ταλαντωτή μέχρι 50MHz.
- Καταστάσεις εξοικονόμησης ενέργειας.
- Ξεχωριστή απενεργοποίηση περιφερειακών για εξοικονόμηση ενέργειας.
- Ο επεξεργαστής ξυπνάει από κατάσταση εξοικονόμησης ενέργειας με εξωτερική διακοπή, USB, RTC(real time clock) και BOD(Brown-out detect).
- Μονής τροφοδοσίας ολοκληρωμένο με κύκλωμα επανεκκίνησης κατά την έναρξη τροφοδοσίας (POR Power-on reset) και BOD. Τάση τροφοδοσίας 3.3V±10% με ανοχή στα 5V των εισόδων/εξόδων I/O.



(1) Pins shared with GPIO.

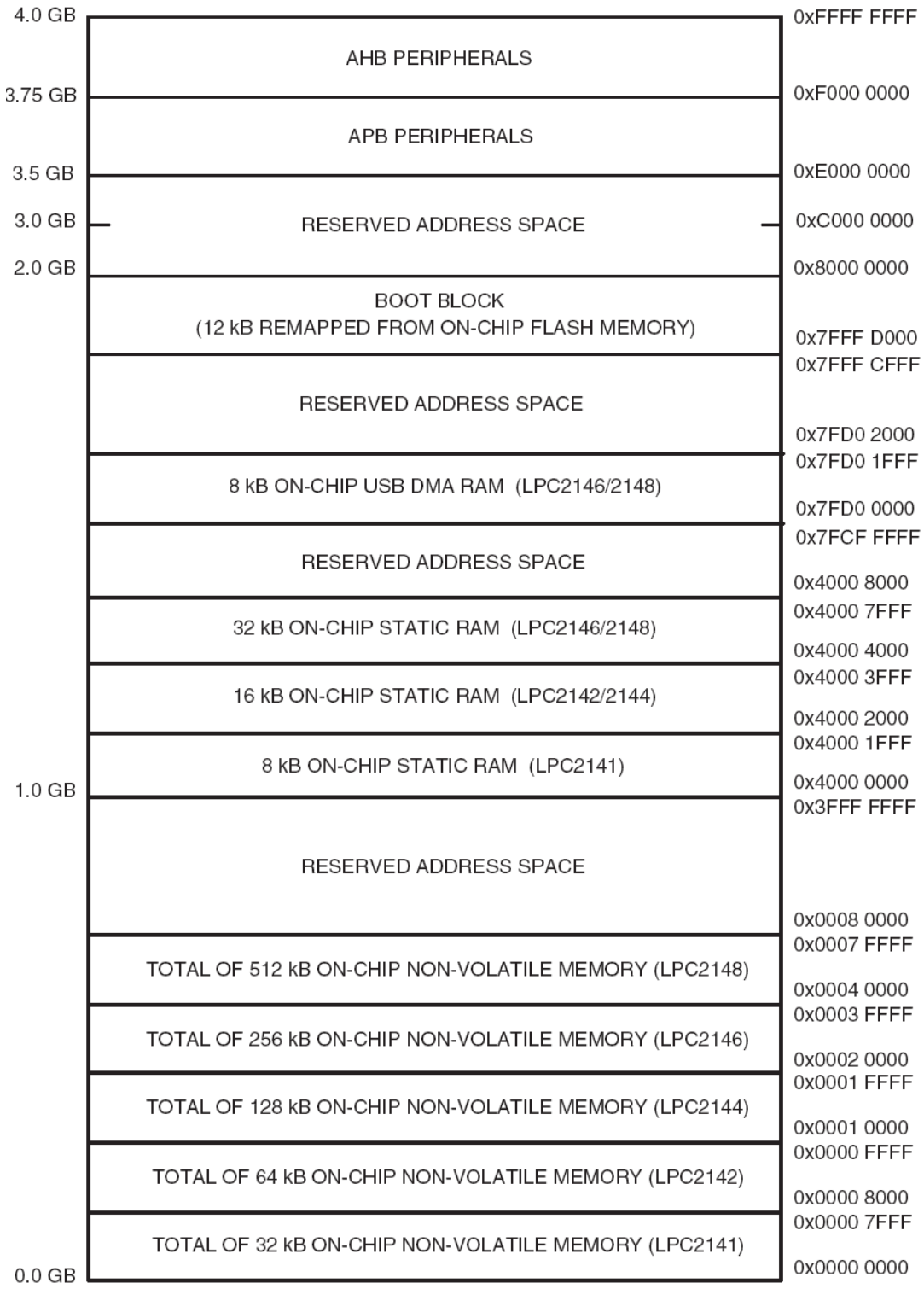
(2) LPCC2144/6/8 only.

(3) USB DMA controller with 8 kB of RAM accessible as general purpose RAM and/or DMA is available in LPC2146/8 only.

(4) LPC2142/4/6/8 only.

Σχήμα 4-1: Μπλοκ διάγραμμα του LPC2148.

Το LPC2148 ενσωματώνει ξεχωριστά τμήματα μνήμης καθώς η λειτουργία του είναι memory mapped. Κάθε περιφερειακό έχει δικές του διευθύνσεις μνήμης για την λειτουργία του.



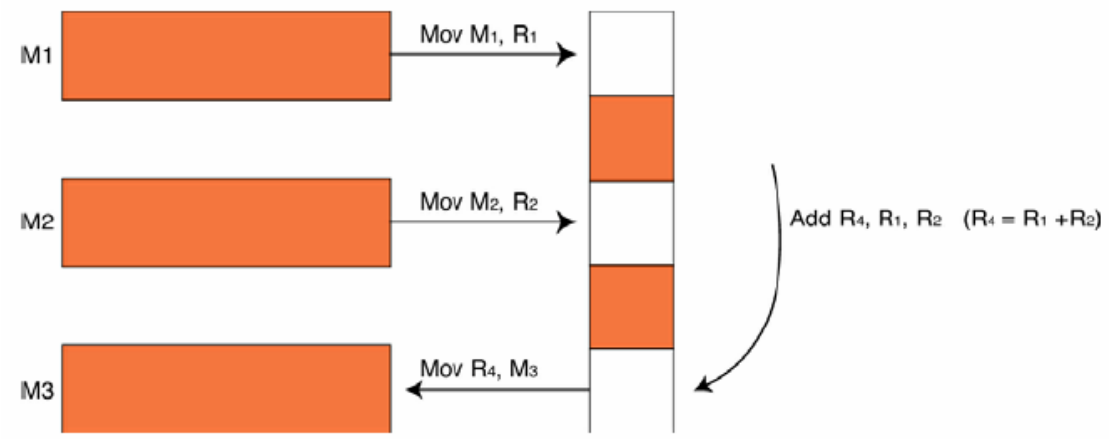
Σχήμα 4-2: Ο χάρτης μνήμης του LPC2148.

4.2 Σωληναγωγός (Pipeline)

Το κύριο χαρακτηριστικό των LPC2000 μικροελεγκτών είναι η αρχιτεκτονική σωληναγωγού (pipeline). Ο σωληναγωγός επεξεργάζεται τις εντολές που παίρνει από την μνήμη. Στους ARM7 ο σωληναγωγός αποτελείται από τρία στάδια. Στο πρώτο στάδιο γίνεται η ανάγνωση της εντολής από την μνήμη, στο δεύτερο γίνεται η αποκωδικοποίηση την εντολής και στο τρίτο η εκτέλεσή της. Αυτή η μορφή της σωλήνωσης είναι η πιο απλή μορφή και έχει ανεξάρτητα στάδια υλικού ώστε όταν εκτελεί μια εντολή να αποκωδικοποιεί μια άλλη και να διαβάζει μια τρίτη εντολή. Με αυτόν τον τρόπο επιταχύνεται η διεκπεραιωτικότητα εντολών της κεντρικής μονάδας επεξεργασίας καθώς και η εκτέλεση των περισσοτέρων ARM εντολών να γίνεται μόνο σε έναν κύκλο ρολογιού.

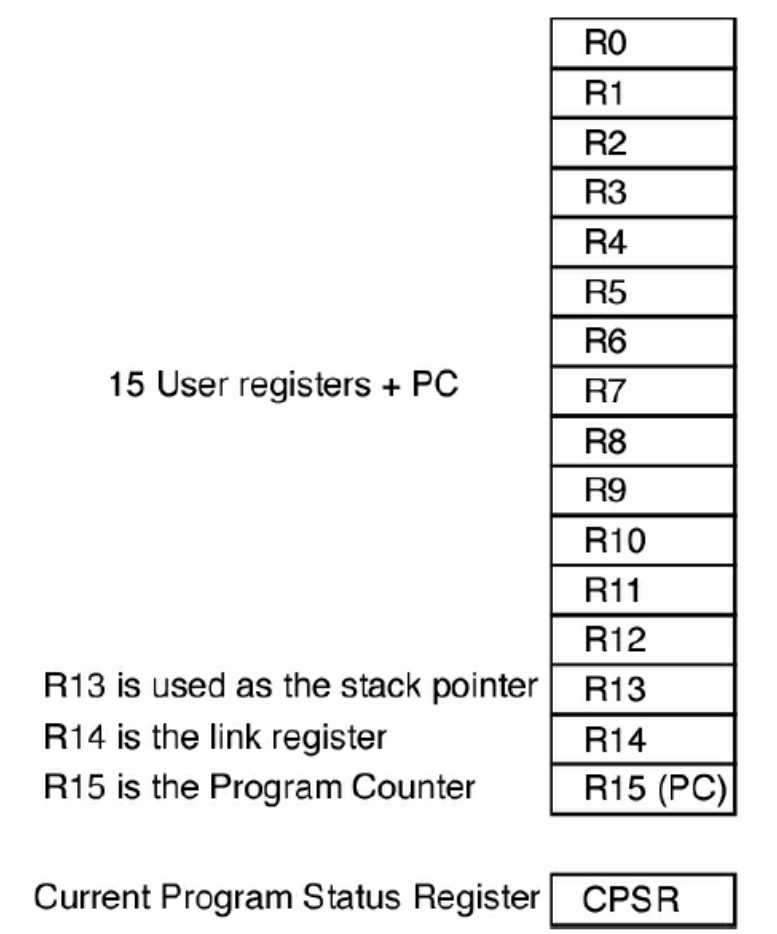
4.3 Καταχωρητές (Registers)

Η αρχιτεκτονική ARM7 καθορίζει ότι για την εκτέλεση εντολών που αφορούν επεξεργασία δεδομένων πρέπει πρώτα τα δεδομένα να μεταφερθούν από την μνήμη που είναι αποθηκευμένα σε κύριους καταχωρητές. Στους καταχωρητές αυτούς γίνεται η επεξεργασία των δεδομένων και μετά τα αποτελέσματα αποθηκεύονται πίσω στην μνήμη.



Σχήμα 4-3: Τρόπος λειτουργίας καταχωρητών για επεξεργασία δεδομένων.

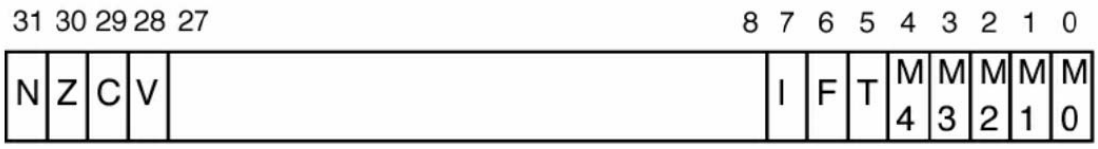
Οι κύριοι αυτοί καταχωρητές είναι οι 16 καταχωρητές χρήστη R0-R15. Καθένας από αυτούς του καταχωρητές έχει μέγεθος 32-bit. Οι R0-R12 δεν έχουν κάποια άλλη καθορισμένη λειτουργία, όμως οι R13-R15 έχουν συγκεκριμένη λειτουργία με τη CPU. Ο R13 χρησιμοποιείται ως δείκτης στοίβας-stack pointer (SP), ο R15 είναι ο μετρητής προγράμματος-program counter (PC) που δείχνει την διεύθυνση της επόμενης εντολής που θα διαβαστεί από τον χώρο εντολών και ο R14 είναι ο καταχωρητής διασύνδεσης-link register (LR). Όταν γίνεται κλήση σε μια συνάρτηση, η διεύθυνση επιστροφής αποθηκεύεται στον καταχωρητή διασύνδεσης. Αυτό επιτρέπει γρήγορη επιστροφή από συναρτήσεις. Όταν όμως η συνάρτηση αυτή καλεί με τη σειρά της άλλες συναρτήσεις τότε ο καταχωρητής διασύνδεσης πρέπει να αποθηκευτεί στην στοίβα (R13).



Σχήμα 4-4: Οι βασικοί καταχωρητές της αρχιτεκτονικής του LPC2148.

4.4 Καταχωρητής προγράμματος

Επιπλέον υπάρχει ένας καταχωρητής που καλείται “καταχωρητής κατάστασης τρέχοντος προγράμματος” (Current Program Status Register) CPSR. Ο καταχωρητής αυτό περιέχει κάποια bit-σημαίες που ελέγχουν και παρουσιάζουν την λειτουργία της CPU.



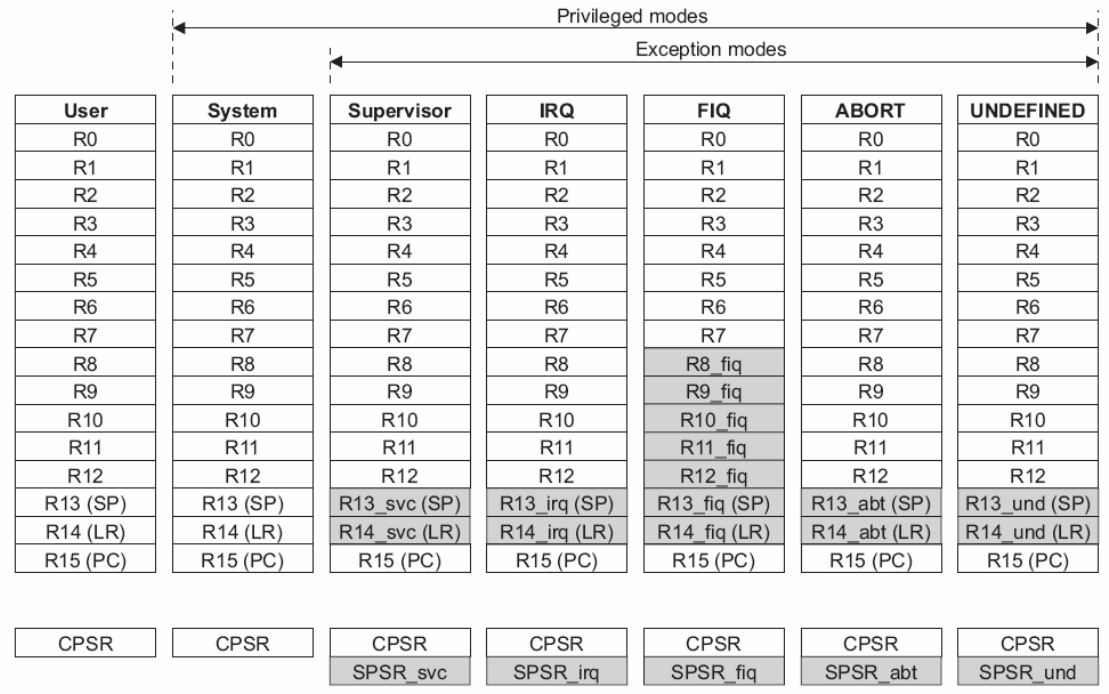
Σχήμα 4-5: Ο καταχωρητής προγράμματος CPSR.

Τα bit 31-28 του CPSR είναι τα N,Z,C,V. Αυτά τα bit δείχνουν αν μια εντολή επεξεργασίας δεδομένων δημιούργησε αποτέλεσμα αρνητικό, μηδενικό, με κρατούμενο ή με υπερχείλιση αντίστοιχα. Τα bit 7 και 8 είναι τα I και F bits. Με αυτά καθορίζουμε πότε είναι ενεργοποιημένες ή απενεργοποιημένες οι

διακοπές του μικροελεγκτή. Το bit 5 είναι το T bit και δείχνει αν οι εντολές που εκτελούνται είναι σε κατάσταση ARM δηλαδή έχουν 32-bit ή σε κατάσταση THUMB δηλαδή έχουν μέγεθος 16-bit για εξοικονόμηση χώρου.

4.5 Καταστάσεις λειτουργίας

Ο ARM7 έχει συνολικά επτά διαφορετικές καταστάσεις λειτουργίας. Ο κώδικας της εφαρμογήςμας συνήθως «τρέχει» σε κατάσταση χρήστη (user mode) με προσπέλαση στους καταχωρητές R0-R15 και στον CPSR. Ωστόσο όταν συμβεί μια εξαίρεση όπως διακοπή, λάθος μνήμης ή διακοπή από το πρόγραμμα ο επεξεργαστής θα αλλάξει κατάσταση λειτουργίας. Όταν γίνει αυτό οι καταχωρητές R0-R12 και R15 μένουν οι ίδιοι αλλά ο R13(LR) και R14(SP) αντικαθίστανται από ένα νέο ζεύγος καταχωρητών, ξεχωριστό για τη συγκεκριμένη κατάσταση λειτουργίας. Αυτό σημαίνει ότι κάθε κατάσταση λειτουργίας έχει την δική του στοίβα και καταχωρητή διασύνδεσης. Επιπλέον η κατάσταση λειτουργίας γρήγορων διακοπών (FIQ) έχει διπλούς καταχωρητές R7-R12. Αυτό σημαίνει ότι μπορεί να γίνει είσοδος σε μια γρήγορη διακοπή χωρίς να χρειάζεται να σωθούν οι καταχωρητές αυτοί στη στοίβα. Αυτό σημαίνει ότι γίνεται αυτόματα εξοικονόμηση χρόνου. Κάθε μία από τις καταστάεις λειτουργίας εκτός από την κατάσταση χρήστη έχει έναν επιπλέον καταχωρητή, τον SPSR (Saved Program Status Register). Αν “τρέχει” η εφαρμογή σε κατάσταση χρήστη όταν γίνει μια εξαίρεση τότε η κατάσταση λειτουργίας θα αλλάξει και το περιεχόμενο του καταχωρητή CPSR θα αποθηκευτεί στον SPSR. Όταν κώδικας της εξαίρεσης θα “τρέξει” και κατά την επιστροφή από την εξαίρεση, το περιεχόμενο του CPSR θα πάρει την παλιά του τιμή που είχε αποθηκευτεί στον SPSR επιτρέποντας στον κώδικα εφαρμογής να επιστρέψει στην εκτέλεσή του.



Σχήμα 4-6: Οι καταστάσεις λειτουργίας του ARM7 και οι καταχωρητές κάθε κατάστασης.

Όταν συμβεί μια εξαίρεση, η CPU θα αλλάξει κατάσταση λειτουργίας και ο μετρητής προγράμματος (PC) θα πάρει μια τιμή που θα δείχνει σε ένα διάνυσμα εξαίρεσης. Ο πίνακας διανυσμάτων εξαίρεσης ξεκινάει από τη διεύθυνση 0 με το διάνυσμα επανεκκίνησης (reset vector) και μετά υπάρχει ένα νέο διάνυσμα εξαίρεσης κάθε τέσσερα bytes. Στην θέση 0x00000014 δεν υπάρχει διάνυσμα εξαίρεσης. Αυτή η θέση κρατήθηκε στον ARM7 για λόγους συμβατότητας με παλαιότερες εκδόσεις των ARM. Ωστόσο στους LPC2000 μικροελεγκτές αυτά τα τέσσερα byte έχουν συγκεκριμένη λειτουργία.

Exception	Mode	Address
Reset	Supervisor	0x00000000
Undefined instruction	Undefined	0x00000004
Software interrupt (SWI)	Supervisor	0x00000008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C
Data Abort (data access memory abort)	Abort	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Σχήμα 4-7: Τα διανύσματα εξαίρεσης του ARM7 και οι διευθύνσεις τους.

Όταν συμβούν ταυτόχρονα διάφορες εξαιρέσεις ακολουθείται η σειρά προτεραιότητας που παρουσιάζεται στο σχήμα 4-8.

Priority	Exception
Highest 1	Reset
2	Data Abort
3	FIQ
4	IRQ
5	Prefetch Abort
Lowest 6	Undefined instruction SWI

Σχήμα 4-8: Οι προτεραιότητες των εξαιρέσεων.

Όταν συμβεί μια εξαίρεση, για παράδειγμα μια εξαίρεση διακοπής (IRQ), ακολουθούνται οι εξής λειτουργίες: Αρχικά η διεύθυνση της επόμενης εντολής που ήταν να εκτελεστεί (PC+4) σώζεται στον καταχωρητή διασύνδεσης (LR). Έπειτα ο καταχωρητής CPSR σώζεται στον SPSR της κατάστασης εξαίρεσης

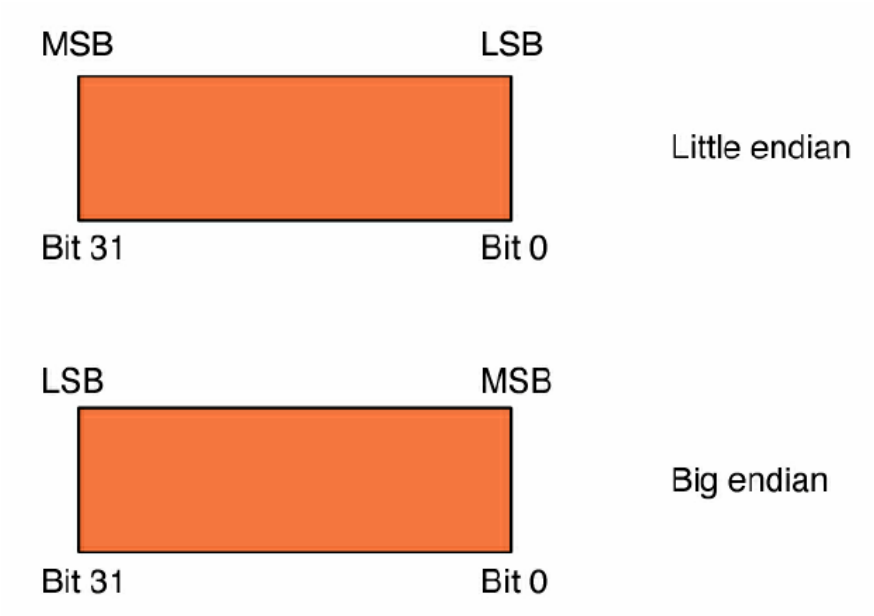
που είναι να πραγματοποιηθεί (πχ. SPSR_irq). Ο PC “φορτώνεται” με την διεύθυνση του διανύσματος εξαίρεσης για την συγκεκριμένη εξαίρεση (εδώ με το διάνυσμα εξαίρεσης διακοπής 0x00000018). Την ίδια στιγμή η κατάσταση λειτουργίας αλλάζει σε IRQ κατάσταση που σημαίνει ότι οι καταχωρητές R13 και R14 αντικαθίστανται από τους IRQ R13 και R14. Κατά την είσοδο στην κατάσταση IRQ, το I bit του CPSR γίνεται ‘1’, απενεργοποιώντας τις διακοπές IRQ. Αν χρειαζόμαστε “φωλιασμένες” διακοπές (nested interrupts), πρέπει μέσω του κώδικα να ενεργοποιήσουμε ξανά τις διακοπές IRQ και να αποθηκεύσουμε τον καταχωρητή διασύνδεσης LR στη στοίβα ώστε να διατηρήθεί η αρχική διεύθυνση επιστροφής. Από το διάνυσμα εξαίρεσης διακοπής, ο κώδικας θα πηδήξει στην ρουτίνα εξυπηρέτησης διακοπής ISR. Το πρώτο πράγμα που πρέπει να κάνει ο κώδικας είναι να σώσει στην IRQ στοίβα όσους από τους R0-R12 καταχωρητές χρησιμοποιεί η ρουτίνα εξυπηρέτησης διακοπής. Μόλις γίνει αυτό μπορεί να αρχίσει η επεξεργασία της εξαίρεσης.

Μόλις ολοκληρωθεί ο κώδικας που επεξεργάζεται την εξαίρεση, πρέπει να επιστρέψει σε κατάσταση χρήστη και να συνεχίσει την λειτουργία από εκεί που την άφησε. Ωστόσο οι εντολές του ARM δεν υποστηρίζουν εντολές “επιστροφή” ή “επιστροφή από διακοπή” και συνεπώς η επεξεργασία του μετρητή εντολών PC πρέπει να γίνει με συνηθισμένες εντολές.

4.6 Το σετ εντολών του ARM7

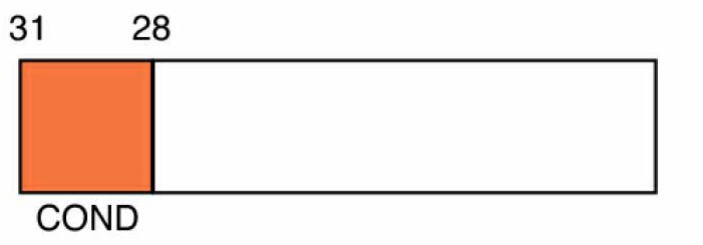
Αν και για τους ισχυρούς μικροελεγκτές όπως ο APM7 ο προγραμματισμός τους γίνεται σε γλώσσα C, είναι σημαντικό να γνωρίζουμε τον κώδικα μηχανής (assembly language) ώστε να μπορούμε να δημιουργούμε αποδοτικά προγράμματα. Ο ARM7 έχει δύο σετ εντολών. Το πρώτο είναι το ARM σετ εντολών στο οποίο κάθε εντολή έχει μέγεθος 32-bit και το THUMB σετ εντολών στο οποίο οι εντολές έχουν μήκος 16-bit.

Οι ARM7 έχουν σχεδιαστεί να λειτουργούν ως big-endian ή little-endian επεξεργαστές. Στους little-endian το μεγαλύτερης σημασίας bit (MSB) της εντολής αντιστοιχεί στο υψηλότερο bit της διεύθυνσης ενώ στους big-endian ισχύει το αντίστροφο. Οι LPC2000 ARM7 της NXP έχουν καθορίστεί ως little-endian. Στον compiler που χρησιμοποιούμε για τα προγράμματά μας πρέπει να έχουμε καθορίσει να μεταγλωττίζει τον κώδικα ως little-endian ώστε να παίρνουμε το σωστό αποτέλεσμα.



Σχήμα 4-9: Εντολές Little endian και Big endian.

Ένα από τα πιο σημαντικά χαρακτηριστικά των ARM εντολών είναι ότι κάθε εντολή μπορεί να εκτελεστεί υπό συνθήκες. Αυτό γίνεται καθώς τα τέσσερα πιο σημαντικά bit του ορίσματος της εντολής συγκρίνονται με τα bit συνθήκης του CPSR. Αν δεν είναι ίδια τότε η εντολή δεν εκτελείται και περνάει ώς εντολή μη-λειτουργίας (no-operation) NOP στη σωλήνωση.



Σχήμα 4-10: Τα bits του τμήματος εκτέλεσης της εντολής υπό συνθήκη.

Έτσι είναι δυνατό να πραγματοποιηθεί μια εντολή επεξεργασίας δεδομένων που επηρεάζει τα bit συνθήκης του CPSR. Στην συνέχεια, ανάλογα με το αποτέλεσμα, οι επόμενες εντολές μπορούν να εκτελεστούν ή να μην εκτελεστούν. Οι βασικές εντολές σε assesmbly όπως οι MOV και ADD μπορούν να πάρουν κάποιο από τα 16 προθέματα συνθηκών που καθορίζουν τις συνθήκες για τις οποίες θα ελεχθούν.

Suffix	Flags	Meaning
EQ	Z set	equal
NE	Z clear	not equal
CS	C set	unsigned higher or same
CC	C clear	unsigned lower
MI	N set	negative
PL	N clear	positive or zero
VS	V set	overflow
VC	V clear	no overflow
HI	C set and Z clear	unsigned higher
LS	C clear and Z set	unsigned lower or same
GE	N equals V	greater or equal
LT	N not equal to V	less than
GT	Z clear AND (N equals V)	greater than
LE	Z set OR (N not equal to V)	less than or equal
AL	(ignored)	always

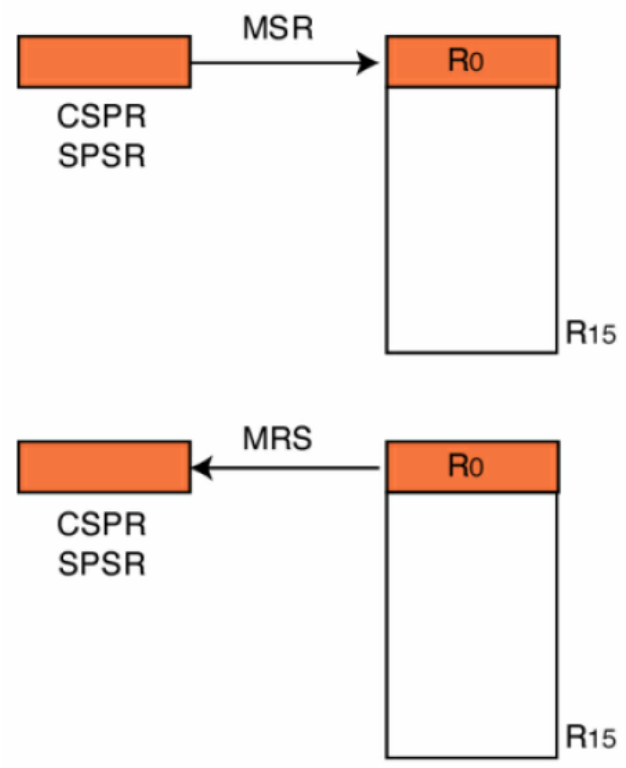
Σχήμα 4-11: Οι συνθήκες εκτέλεσης εντολής.

Για παράδειγμα η εντολή:
EQMOV R1, #0x00800000
Θα μεταφέρει το 0x00800000 στον καταχωρητή R1 μόνο αν η τελευταία εντολή επεξεργασίας δεδομένων έθεσε το Z bit-σημαία του CPSR.

Αυτή η μέθοδος γίνεται για να βελτιώσει την απόδοση του επεξεργαστή καθώς κάθε φορά που υπάρχει εντολή διακλάδωσης (branch) ή άλματος (jump) η σωλήνωση πρέπει να αδειάζει και αυτό προκαλεί μείωση στην συνολική απόδοση του επεξεργαστή.

Οι βασικές ομάδες του σετ εντολών ARM ανήκουν στις εξής έξι κατηγορίες: διακλάδωσης (branching), επεξεργασία δεδομένων (data processing), μεταφορά δεδομένων (data transfer), μεταφορά μπλοκ (block transfer), πολλαπλασιασμός (multiply) και διακοπή προγράμματος (software interrupt).

Οι CPSR και SPSR είναι καταχωρητές του επεξεργαστή αλλά δεν ανήκουν στους κύριους καταχωρητές. Μόνο δύο εντολές ARM μπορούν να λειτουργήσουν σε αυτούς τους καταχωρητές άμεσα. Οι MSR και MRS εντολές μεταφέρουν τους CPSR και SPSR από και προς συγκεκριμένους καταχωρητές. Για παράδειγμα αν θέλουμε να απενεργοποιήσουμε τις IRQ διακοπές, το περιεχόμενο του CPSR πρέπει να μεταφερθεί σε έναν καταχωρητή, το “I” bit πρέπει να τεθεί για να απενεργοποιηθούν οι διακοπές και μετά ο CPSR πρέπει να επαναπρογραμματιστεί.

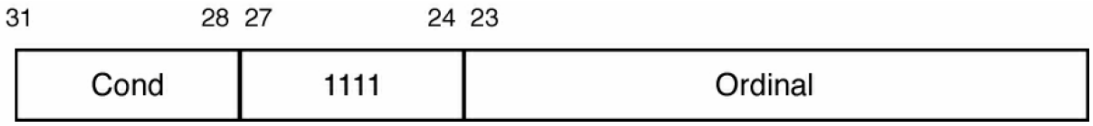


Σχήμα 4-12: Εντολές αλλαγών των CPSR και SPSR καταχωρητών.

Οι MSR και MRS εντολές λειτουργούν σε όλες τις καταστάσεις λειτουργίας του επεξεργαστή εκτός από την κατάσταση χρήστη. Μόλις ο επεξεργαστής μπει σε κατάσταση χρήστη ο μόνος τρόπος για να βγει είναι μέσω μιας εξαίρεσης, επανεκκίνησης, διακοπής, γρήγορης διακοπής ή διακοπή προγράμματος.

4.7 Διακοπή προγράμματος

Η εντολή διακοπής προγράμματος (SWI) δημιουργεί μια εξαίρεση κατά την εκτέλεση, υποχρεώνει τον επεξεργαστή να μπει σε κατάσταση επιβλέπων (supervisor mode) και φορτώνει στον μετρητή προγράμματος PC την 0x00000008. Όπως με τις υπόλοιπες εντολές ARM, η εντολή SWI περιέχει στα τέσσερα πιο σημαντικά bits τους κωδικούς συνθήκης. Έπειτα ακολουθεί ο κωδικός ορίσματος. Το υπόλοιπο της εντολής είναι άδειο. Ωστόσο είναι δυνατό να εισάγουμε έναν κωδικό σε αυτά τα αχρησιμοποίητα bit. Καθώς ο επεξεργαστής μπαίνει σε διακοπή προγράμματος, ο κώδικας μπορεί να εξετάζει αυτά τα bit και να επιλέγει ποιον κώδικα να τρέξει. Έτσι είναι δυνατό να χρησιμοποιηθεί η εντολή SWI ώστε να γίνονται κλησεις σε προστατευμένη κατάσταση, να τρέχει προνομιακός κώδικας και να γίνονται κλήσεις λειτουργικού συστήματος.



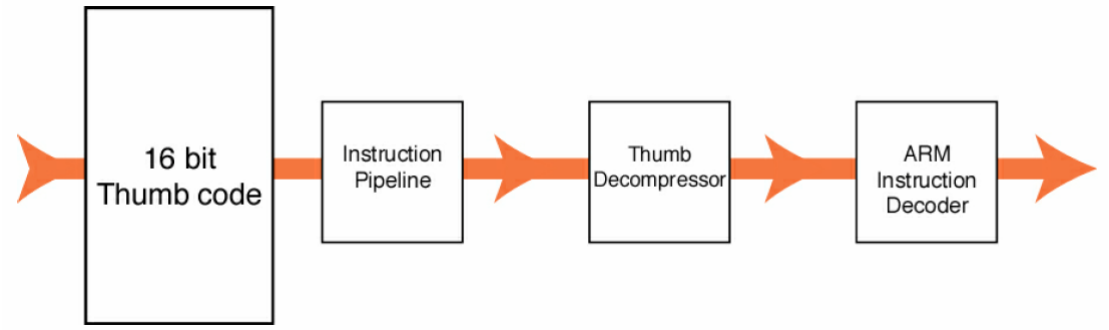
Σχήμα 4-13: Εντολή SWI.

4.8 Η μονάδα MAC

Ο ARM7 έχει μια μονάδα πολλαπλασιαστή τύπου συσσωρευτή (Multiply Accumulate Unit) MAC. Η μονάδα αυτή υποστηρίζει πολλαπλασιασμό ακεραίων (integer) και μεγάλων ακεραίων (long interger) αριθμών. Οι εντολές πολλαπλασιασμού ακεραίων, πολλαπλασιάζουν δύο 32-bit αριθμούς και αποθηκεύουν το αποτέλεσμα σε έναν καταχωρητή 32-bit (modulo32). Μια εντολή πολλαπλασιασμού συσσώρευσης θα πάρει αυτό το αποτέλεσμα και θα το προσθέσει σε ένα τρέχων άθροισμα. Ο πολλαπλασιασμός μεγάλων δεκαδικών αριθμών πολλαπλασιάζει δύο 32-bit αριθμούς και το αποτέλεσμα αποθηκεύεται σε δύο 32-bit καταχωρητές. Ομοίως υπάρχει εντολή πολλαπλασιασμού με συσσώρευση μεγάλων ακεραίων.

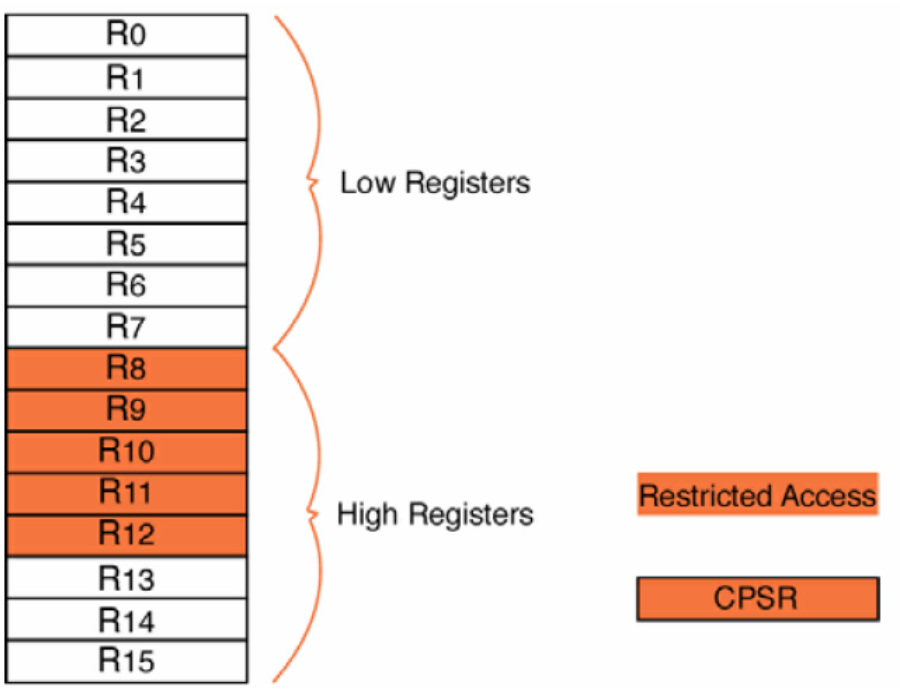
4.9 Το σετ εντολών Thumb

Αν και ο ARM7 είναι ένας 32-bit επεξεργαστής, έχει ένα δεύτερο σετ εντολών 16-bit σε μήκος που καλείται THUMB το οποίο είναι στην πραγματικότητα ένα συμπιεσμένο σετ ARM εντολών. Αυτό επιτρέπει τις εντολές να αποθηκευτούν σε 16-bit μορφή, να επεκταθούν σε 32-bit ARM εντολές και μετά να εκτελεσθούν. Αν και οι εντολές THUMB μειώνουν την απόδοση του κώδικα σε σύγκριση με τις εντολές ARM, πετυχαίνουν μεγαλύτερη συμπύκνωση κώδικα. Επίσης είναι σημαντικό να γίνεται ταυτόχρονη χρήση σε ένα πρόγραμμα και των δύο σετ εντολών όπου είναι δυνατό ώστε να πετυχαίνουμε και μείωση του χρόνου εκτέλεσης και του μεγέθους του κώδικα. Μεταγλωττίζοντας τον κώδικα σε κατάσταση THUMB είναι δυνατό να πετύχουμε μείωση μεγέθους κώδικα 30% ένω με μεταγλώττιση σε κατάσταση ARM, ο κώδικας τρέχει 40% πιο γρήγορα.



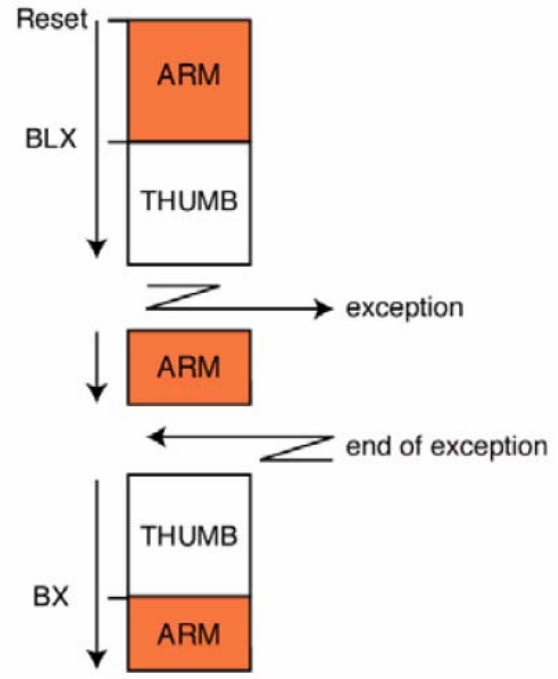
Σχήμα 4-14: Αποκωδικοποίηση εντολής THUMB.

Το σετ εντολών THUMB μοιάζει πιο πολύ με το τυπικό σετ εντολών των μικροελεγκτών. Δεν έχουν εκτέλεση υπό συνθήκες (εκτός από τις εντολές διακλάδωσης υπό συνθήκη) και οι εντολές επεξεργασίας δεδομένων έχουν μορφή δύο διευθύνσεων στην οποία ο καταχωρητής προορισμού είναι ένας από τους καταχωρητές προέλευσης. Επιπλέον οι εντολές THUMB δεν έχουν προσβαση σε όλους τους καταχωρητές. Οι εντολές επεξεργασίας δεδομένων περιορίζονται στους καταχωρητές R0-R7.



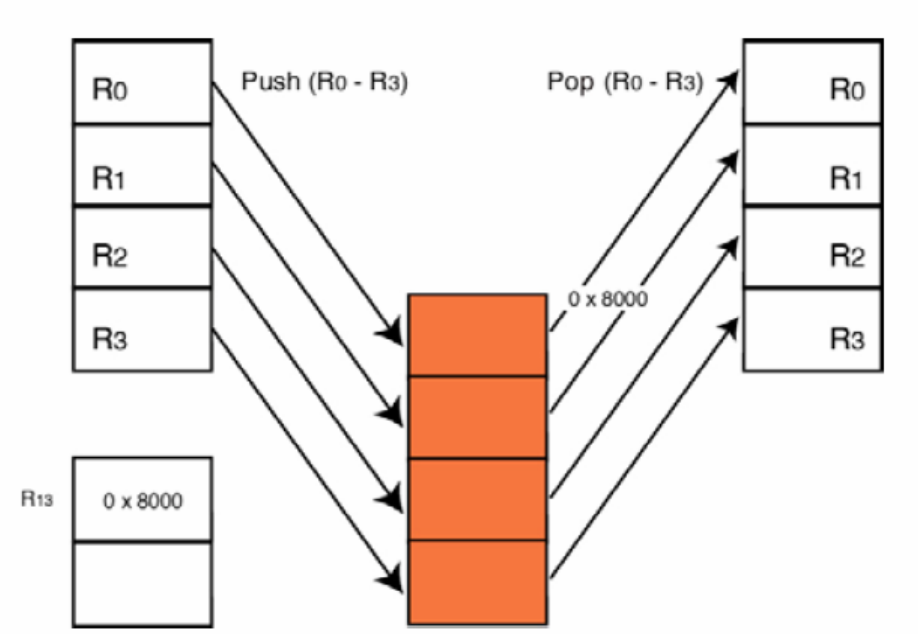
Σχήμα 4-15: Περιορισμοί προσπέλασης στους καταχωρητές.

Η προσπέλαση στους R8-R12 περιορίζεται σε λίγες εντολές : MOV,ADD και CMP. Δεν υπάρχει πρόσβαση στους CPSR και SPSR και έτσι πρέπει πρώτα να αλλάζει η λειτουργία σε ARM κατάσταση. Μπορούμε να αλλάξουμε καταστάσεις με τις εντολες BX και BLX. Επίσης μετά από επανεκκίνηση (reset) ή όταν ο επεξεργαστής εισέρχεται σε κατάσταση εξαίρεσης θα λειτουργεί σε ARM κατάσταση.



Σχήμα 4-16: Αλλαγή σε τρόπο λειτουργίας ARM σε κατάσταση εξαίρεσης.

Ακόμα το THUMB σετ εντολών έχει πιο παραδοσιακές PUSH και POP εντολές για λειτουργία στοίβας και χρησιμοποιεί τον R13 για δείκτη στοίβας. Τέλος υπάρχει και εντολή διακοπής προγράμματος SWI που λειτουργεί όπως η εντολή του ARM σετ εντολών αλλά έχει μόνο 8 αχρησιμοποίητα bits που μπορούβ να δώσουν μέχρι 255 SWI κλήσεις.



Σχήμα 4-17: Εντολές Push και Pop σε τρόπο λειτουργίας Thumb.

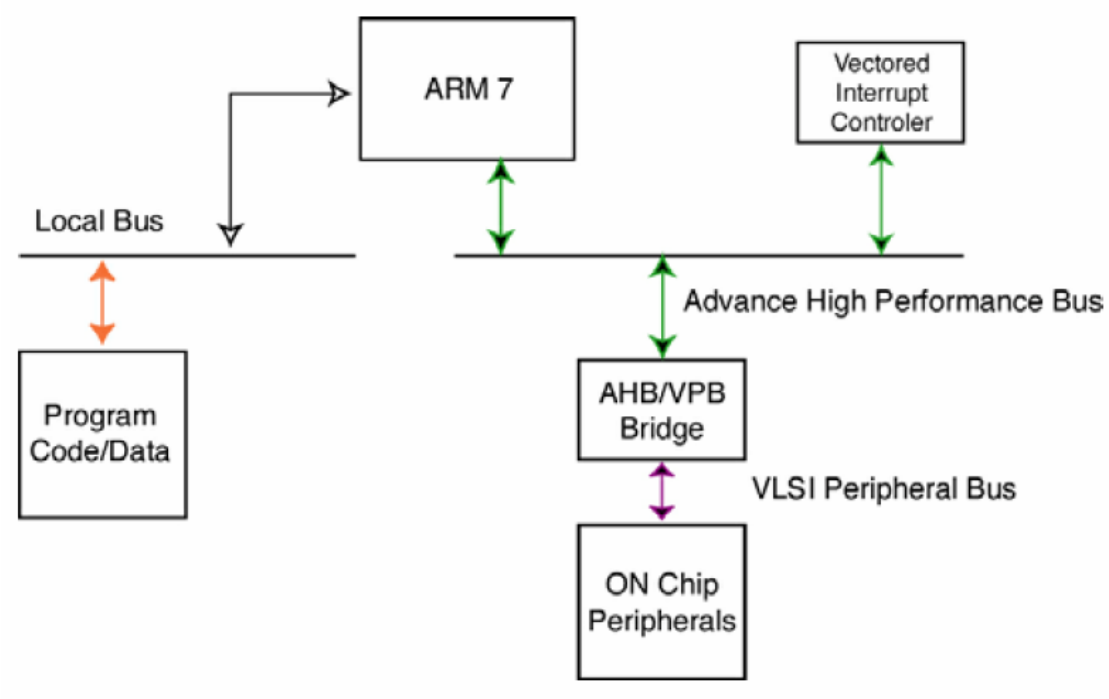
4.10 Η αρχιτεκτονική διαδρόμου (Bus structure)

Για τον προγραμματιστή, η μνήμη όλων των LPC2100 μικροελεγκτών είναι ένα πεδίο από παραπλήσιες 32-bit διευθύνσεις. Ωστόσο, ο ίδιος ο μικροελεγκτής αποτελείται από διάφορους διαδρόμους (buses). Ο ARM7 πυρήνας είναι συνδεδεμένος με τον Προχωρημένο Υψηλής Απόδοσης Διάδρομο (Advanced High Performance Bus) AHB που έχει καθοριστεί για τους ARM. Είναι ο πιο γρήγορος τρόπος σύνδεσης περιφερειακών συσκευών στον ARM7 πυρήνα. Συνδεδεμένα με τον AHB είναι ο ελεγκτής διανυσμάτων διακοπών και μια γέφυρα προς τον δεύτερο διάδρομο που ονομάζεται VLSI διάδρομος περιφερειακών (VLSI peripheral bus) VPB. Αφού ελεγκτής διανυσμάτων διακοπών είναι υπεύθυνος για να ελέγχει όλες τις πηγές διακοπών, είναι δυνδεδεμένος στον ARM7 πυρήνα με τον πιο γρήγορο διάδρομο.

Τα υπόλοιπα περιφερειακά είναι συνδεδεμένα στον VPB. Ο διάδρομος VPB έχει ένα διαιρέτη ρολογιού για να μπορεί να τρέχει σε χαμηλότερες ταχύτητες από τον AHB και τον ARM7 πυρήνα. Αυτό είναι χρήσιμο για δύο λόγους. Πρώτα μπορούμε να τρέχουμε τα περιφερειακά χρήστη σε χαμηλότερη ταχύτητα και με αυτό τον τρόπο να εξοικονομούμε ενέργεια. Επίσης δίνει την δυνατότητα να υπάρχουν πιο αργά περιφερειακά χωρίς να δημιουργείται καθυστέρηση (bottleneck) στον AHB. Ωστόσο στους LPC2000 μικροελεγκτές

όλα τα περιφερειακά μπορούν να τρέχουν με την μέγιστη ταχύτητα στα 60MHz, συνεπώς ο VPB μπορεί να καθοριστεί να έχει την ίδια ταχύτητα με τον AHB. Μετά από επανεκκίνηση του μικροελεγκτή ο VPB διαιρέτης είναι ρυθμισμένος να διαιρεί το AHB ρολόι με το τέσσερα ώστε όλα τα περιφερειακά πάνω στο ολοκληρωμένο να τρέχουν με ταχύτητα ¼ του ρολογιού της CPU.

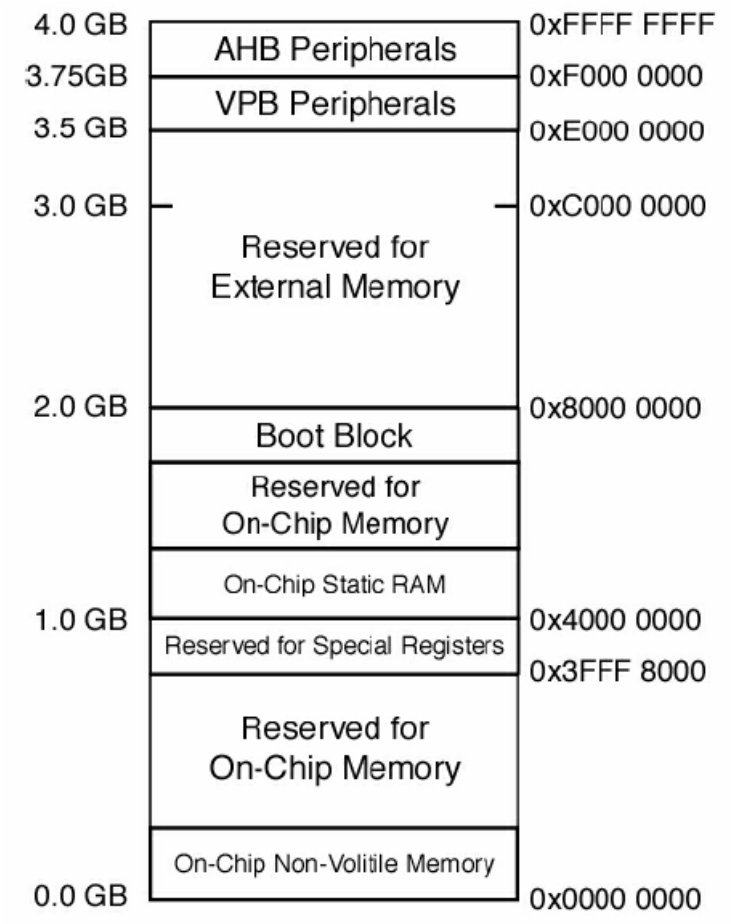
Τέλος υπάρχει ένας τρίτος τοπικός διάδρομος ο οποίος συνδέει την εσωτερική Flash και RAM μνήμη του ολοκληρωμένου με την CPU.



Σχήμα 4-18: Οι διάδρομοι της ARM7 αρχιτεκτονικής.

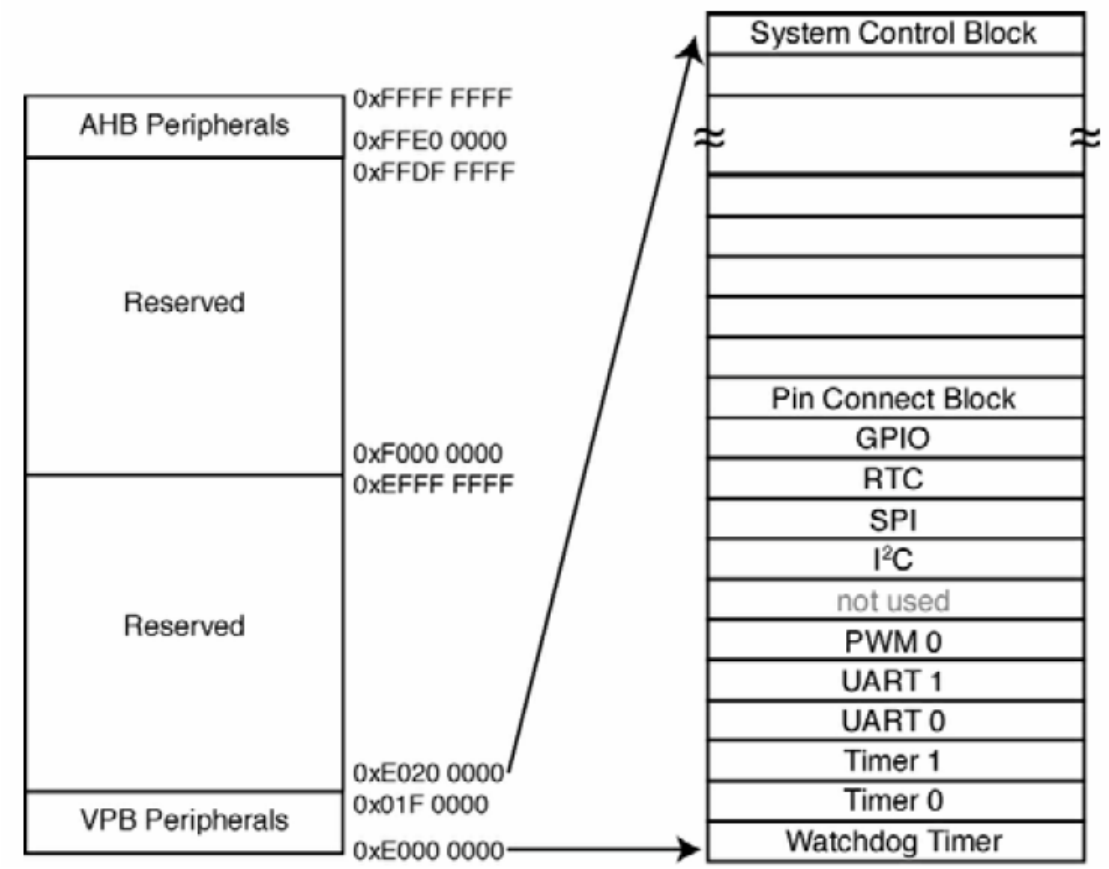
4.11 Ο χάρτης μνήμης (Memory Map)

Ανεξάρτητα από τους διάφορους εσωτερικούς διαδρόμους, οι LPC2000 έχουν τελείως γραμμικό χάρτη μνήμης.



Σχήμα 4-19: Ο χάρτης μνήμης των LPC2000.

Η εσωτερική μνήμη Flash ξεκινάει από την διεύθυνση 0x00000000 και αυξάνεται ενώ η εσωτερική RAM ξεκινάει από την 0x40000000 και αυξάνεται. Οι LPC2000 μικροελεγκτές έχουν προγραμματιστεί κατά την κατασκευή τους με Flash Bootloader και το ARM real time monitor πρόγραμμα. Αυτά τα προγράμματα έχουν τοποθετηθεί μεταξύ των διευθύνσεων 0x7FFFFFFF και 0x80000000. Η περιοχή μεταξύ 0x80000000 και 0xE0000000 έχει δεσμευτεί για εξωτερική μνήμη (κυρίως για τους LPC22xx μικροελεγκτές).

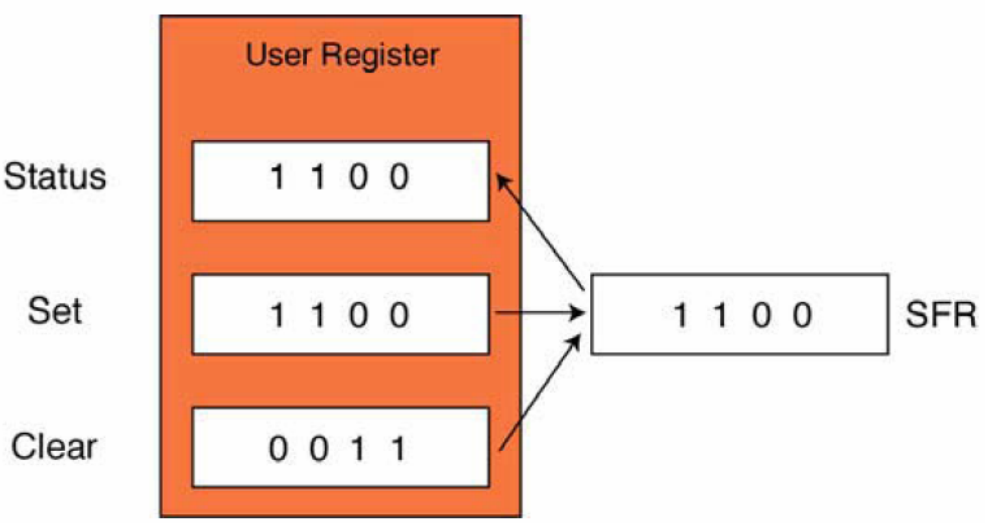


Σχήμα 4-20: Ο χάρτης μνήμης των περιφερειακών.

Τα περιφερειακά χρήστη που συνδέονται με τον VPB, βρίσκονται μεταξύ των διευθύνσεων 0xE0000000 - 0xE0200000 και σε κάθε περιφερειακό έχει κατανεμηθεί μια σελίδα μνήμης 16K. Τέλος η μονάδα διανυσμάτων διακοπών βρίσκεται στην κορυφή στην διεύθυνση 0xFFFFF000. Αν ο κώδικας χρήστη προσπαθήσει να προσπελάσει μνήμη έξω από αυτές τις περιοχές ή μη υπάρχουσα μνήμη μέσα σε αυτές, η CPU δημιουργεί εξαίρεση ματαίωσης (abort exception).

4.12 Προγραμματισμός καταχωρητών

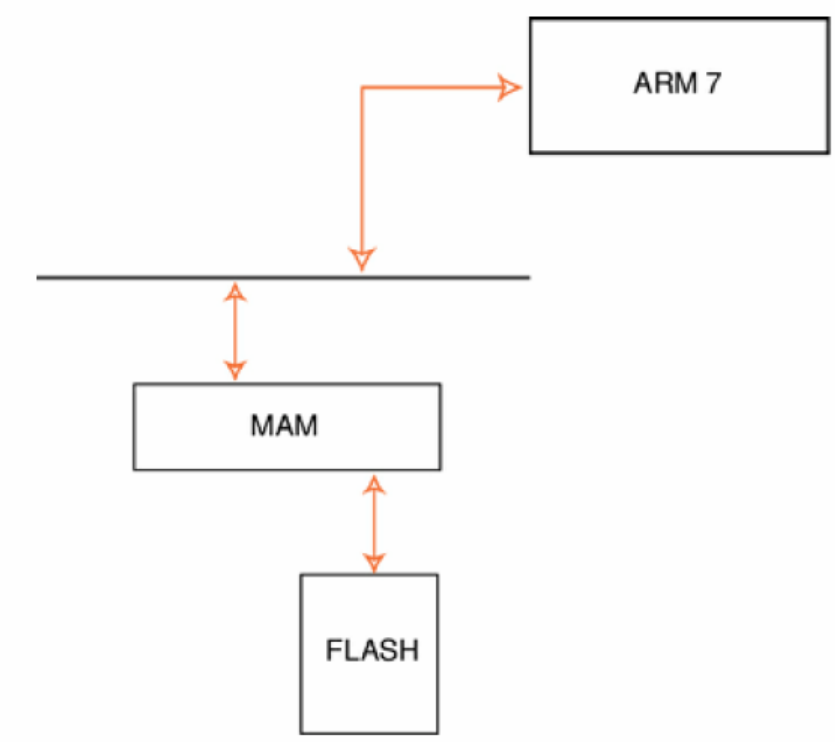
Κάθε καταχωρητής ειδικής λειτουργίας (Special function register) SFR ελέγχεται μέσω τριών καταχωρητών. Έναν Set καταχωρητή ο οποίος χρησιμοποιείται για να θέτει bits, έναν Clear καταχωρητή για να μηδενίζει bits και έναν Status καταχωρητή για να διαβάζει το περιεχόμενο των SFR.



Σχήμα 4-21: Προγραμματισμός καταχωρητών.

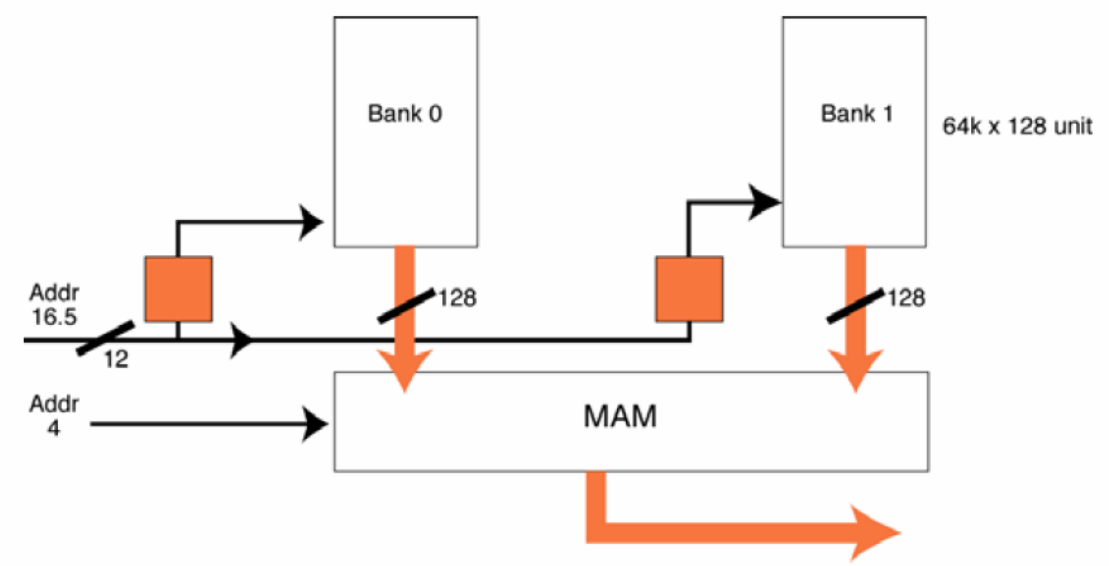
4.13 Μονάδα επιτάχυνσης μνήμης (MAM)

Η μονάδα επιτάχυνσης μνήμης (Memory acceleration module) MAM είναι ο κύριος λόγος υψηλής ταχύτητας εκτέλεσης εντολών στους μικροελγκτές LPC2100. Συνδέεται στον τοπικό διάδρομο και βρίσκεται μεταξύ της μνήμης Flash και της ARM7 CPU.



Σχήμα 4-22: Η μονάδα επιτάχυνσης μνήμης παρεμβάλεται μεταξύ του επεξεργαστή και της μνήμης Flash.

Ένας από τους σημαντικούς περιορισμούς στην σχεδίαση ένος υψηλής απόδοσης μικροελεγκτή βασισμένο στην ARM7 CPU είναι ο χρόνος προσπέλασης στην μνήμη Flash. Η ARM7 CPU είναι ικανή να τρέχει έως τα 80MHz ωστόσο η εσωτερική μνήμη Flash έχει χρόνο προσπέλασης 50ns. Συνεπώς, αν τρέχαμε κώδικα κατευθείαν από την Flash θα περιοριζόταν η ταχύτητα της CPU στα 20MHz. Υπάρχουν διάφοροι τρόποι για να λυθεί αυτό το πρόβλημα. Ο πιο απλό είναι να φορτώσουμε τα πιο κρίσιμα μέρη του κώδικά μας στην μνήμη RAM του μικροελεγκτή και να τα τρέχουμε από εκεί. Η μνήμη RAM έχει πολύ πιο μικρό χρόνο προσπέλασης από την Flash συνεπώς η συνολική απόδοση θα αυξηθεί. Ωστόσο το αρνητικό είναι ότι η μνήμη RAM έχει πολύ μικρότερο μέγεθος από την Flash με αποτέλεσμα να πρέπει να τρέχουμε μικρό κώδικα το οποίο περιορίζει τις δυνατότητες των εφαρμογών μας. Μια άλλη λύση είναι η χρήση κρυφής μνήμης cashe η οποία όμως δεν έχει εφαρμοστεί στους ARM7 διότι είναι πολύπλοκο κύκλωμα και ξεφεύγει από την φιλοσοφία των ARM7. Η μονάδα επιτάχυνσης μνήμης MAM επιτρέπει άμεση επικοινωνία με την Flash του μικροελεγκτή παραμένοντας ωστόσο αποδοτική.

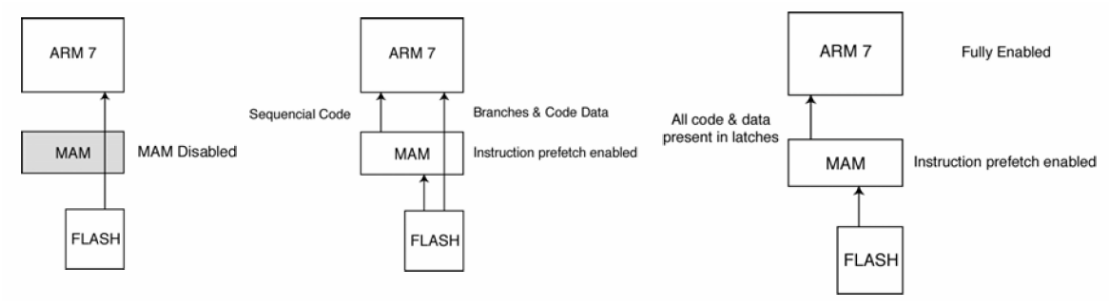


Σχήμα 4-23: Η λειτουργία της MAM.

Όπως στην cashe, η MAM προσπαθεί να έχει την επόμενη ARM εντολή στην τοπική μνήμη ώστε να μπορεί να εκτελεστεί σε έγκυρο χρόνο από τον επεξεργαστή. Αρχικά η μνήμη Flash χωρίζεται σε δύο μέρη τα οποία έχουν μέγεθος 128 bits και μπορούν να προσπελαστούν ανεξάρτητα το ένα από το άλλο. Αυτό σημαίνει ότι σε μία προσπέλαση στην Flash μπορούν να φορτωθούν 4 εντολές ARM ή 8 εντολές THUMB. Ο κώδικας χρήστη έχει διαστρωματωθεί μεταξύ των δύο μερών Flash, έτσι κατά τη διάρκεια εκτέλεσης διαδοχικού κώδικα, ο κώδικας που έχει μεταφερθεί από το ένα μέρος της Flash στην MAM εκτελείται ενώ τα υπόλοιπα 128 bits εντολών από το δεύτερο μέρος προετοιμάζονται. Αυτό εξασφαλίζει ότι ο κώδικας του δεύτερου μέρους θα είναι έτοιμος για εκτέλεση όταν εκτελεστουν και τα 128 bit του πρώτου μέρους. Αυτή η τεχνική δουλεύει πολύ καλά με τις εντολές ARM που μπορούν να χρησιμοποιήσουν τους κώδικες συνθηκών για να εξομαλύνουν μικρές διακλαδώσεις στον κώδικα ώστε να κρατήσουν την ροή

του κώδικα γραμμική. Στη περίπτωση μικρών βρόχων και αλμάτων η MAM έχει μονάδες προσωρινής αποθήκευσης που κρατούν προσφάτως φορτωμένες εντολές που μπορούν να ξαναεκτελεστούν αν χρειαστεί.

Η λειτουργία της MAM είναι διαυγής στον χρήστη και καθορίζεται από δύο καταχωρητές, τον καταχωρητή χρονισμού και τον καταχωρητή ελέγχου. Υπάρχουν επίσης κάποιοι καταχωρητές που παρέχουν πληροφορίες για την λειτουργία της MAM. Ο καταχωρητής χρονισμού χρησιμοποιείται για να ελέγξει την σχέση μεταξύ του ρολογιού του επεξεργαστή και του χρόνου προσπέλασης της Flash. Γράφοντας στα πρώτα τρία bits του καταχωρητή χρονισμού καθορίζουμε τον αριθμό κύκλων του επεξεργαστή που χρειάζονται στην MAM για να προσπελάσει την Flash. Καθώς η Flash έχει χρόνο προσπέλασης 20MHz και ο επεξεργαστής έχει μέγιστο ρολόι στα 60MHz, ο αριθμός κύκλων που χρειάζονται για να προσπελαστεί η Flash είναι τρεις. Έτσι κάθε τρεις κύκλους ρολογιού του επεξεργαστή μπορούμε να φορτώνουμε τέσσερις εντολές στην MAM ώστε να έχουμε καλή απόδοση του συστήματος. Ο καταχωρητής ελέγχου ελέγχει τον τρόπο λειτουργία της MAM.



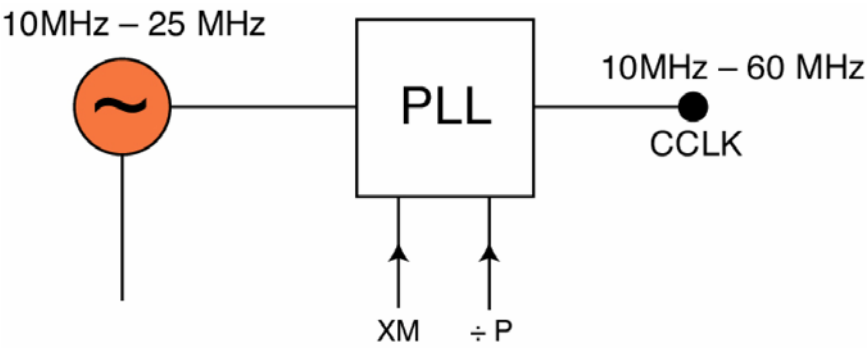
Σχήμα 4-24: Διάφοροι τρόποι λειτουργίας MAM.

Κατά την επανεκκίνηση, η MAM είναι απενεργοποιημένη και η πρόσβαση στον κώδικα γίνεται κατευθείαν από την μνήμη Flash. Είναι δυνατόν η μερική ενεργοποίηση της MAM ώστε όλες οι διαδοχικές εντολές να μεταφέρονται από αυτή, αλλά οι διακλαδώσεις και τα δεδομένα που αποτελούν σταθερές να μεταφέρονται κατευθείαν από την Flash. Επίσης η MAM μπορεί να ενεργοποιηθεί πλήρως ώστε να φέρνει όλες τις εντολές από την Flash και στην συνέχεια να τις τροφοδοτεί στον επεξεργαστή.

4.14 Βρόχος κλειδώματος φάσης (PLL)

Το (Phased Locked Loop) PLL χρησιμοποιείται για να μετατρέψει μια συχνότητα ταλάντωσης μεταξύ 10MHz και 25MHz που παίρνει από έναν εξωτερικό κρύσταλλο σε μια μεγαλύτερη συχνότητα έως 60MHz πολλαπλασιάζοντάς την. Η συχνότητα που προκύπτει χρησιμοποείται από τον επεξεργαστή και τα περιφερειακά του ARM7. Αυτό επιτρέπει στους LPC2000 μικροελεγκτές να τρέχουν από έναν χαμηλότερης συχνότητας ταλαντωτή και με αυτόν τον τρόπο να μειώνουν τις ηλεκτρομαγνητικές

εκπομπές (EMC). Επίσης το PLL καθαρίζει την συχνότητα εισόδου από τον θόρυβο και δημιουργεί μια καλύτερη συχνότητα ταλάντωσης για τον μικροελεγκτή. Τέλος η συχνότητα που παράγει το PLL μπορεί να αλλάξει δυναμικά, επιτρέποντας στην συσκευή να μειώσει τον ρυθμό λειτουργία της όταν βρίσκεται σε ηρεμία ώστε να καταναλώνει λιγότερη ενέργεια.



Σχήμα 4-25: Το PLL του LPC2148.

Μέσα στο PLL υπάρχουν δύο σταθερές που πρέπει να προγραμματιστούν ώστε να καθοριστεί το ρολόι της CPU και του AHB διαδρόμου. Αυτό το ρολόι ονομάζεται Cclk. Η πρώτη σταθερά πολλαπλασιάζει την συχνότητα του εξωτερικού κρυστάλλου. Η συχνότητα εξόδου του PLL δίνεται από την σχέση:

Cclk = M x Osc

Στον βρόχο ανάδρασης του PLL υπάρχει ένας ταλαντωτής που ελέγχεται από ρεύμα (current controlled oscillator) CCO που πρέπει να λειτουργεί μεταξύ 156MHz και 320MHz. Η δεύτερη σταθερά προγραμματίζεται ώστε να εξασφαλίζει την λειτουργία αυτού του CCO στα όρια των προδιαγραφών του. Η συχνότητα λειτουργίας του CCO καθορίζεται ως εξής:

Fcco = Cclk x 2 x P

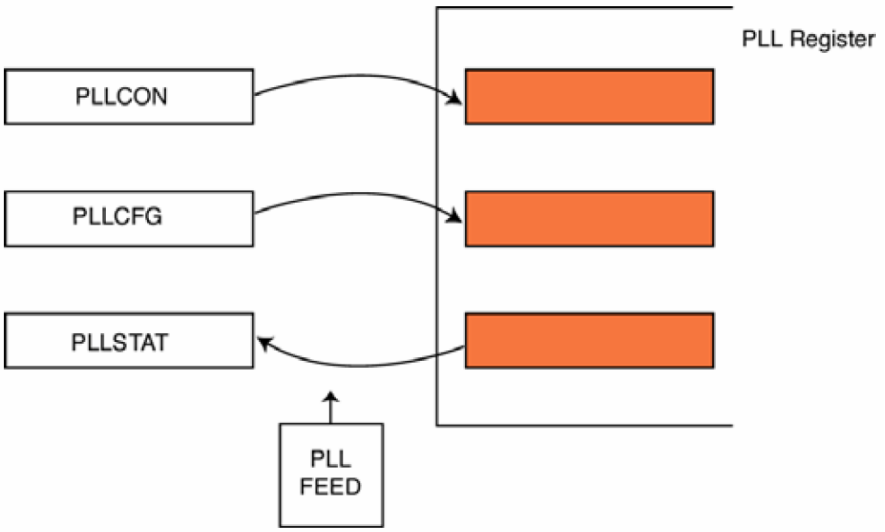
Στο κύκλωμα που έχουμε κατασκευάσει, έχουμε εξωτερικό κρύσταλλο συχνότητας 12MHz. Για να πετύχουμε την μέγιστη δυνατή συχνότητα λειτουργία των 60MHz πρέπει το M να πάρει την τιμή 5.

M = Cclk / Osc = 60 / 12 = 5

Για το P έχουμε :

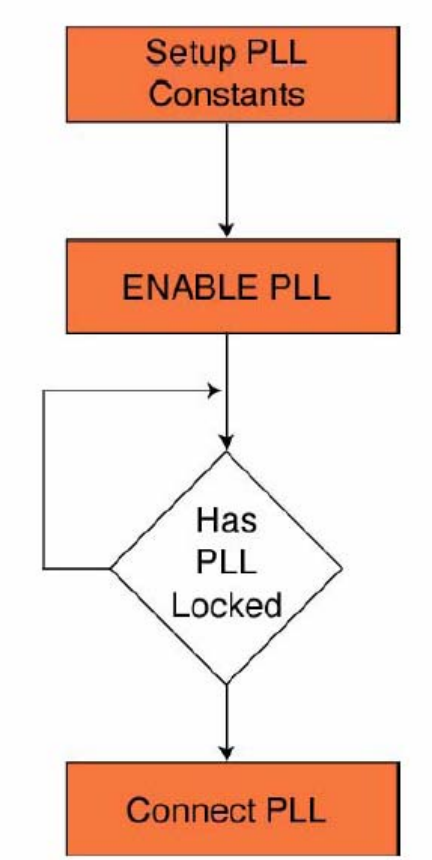
156 < Fcco < 320 δηλαδή 156 < 60 x 2 x P < 320

Από τα παραπάνω προκύπτει P=2.



Σχήμα 4-26: Διάγραμμα ελέγχου PLL.

Οι τιμές που γράφονται στους SFR καταχωρητες χρήστη δεν μεταφέρονται στους καταχωρητές του PLL αν δεν εγγραφεί μια ακολουθία τροφοδότησης στον καταχωρητή τροφοδότησης του PLL. Αφού ενημερώσουμε τους καταχωρητές PLLCON και PLLCFG πρέπει να γράψουμε 0x000000AA και αμέσως μετά 0x00000055 στον PLLFEED καταχωρητή. Αυτές οι τιμές πρέπει να εγγραφούν σε διαδοχικούς κύκλους. Αυτός είναι ο λόγος που πρέπει να απενεργοποιούμε πρώτα τις διακοπές πρίν τροφοδοτήσουμε τον PLLFEED με την ακολουθία. Για να ρυθμίσουμε το PLL πρέπει να γράψουμε τις τιμές των M και P στον PLLCFG καταχωρητή. Ύστερα χρησιμοποιώντας τον PLLCON καταχωρητή το PLL ενεργοποιείται. Αυτός εκκινεί το PLL ωστόσο χρειάζεται ένα χρονικό διάστημα έναρξης πριν σταθεροποιηθεί η συχνότητα ώστε να είναι αρκετά σταθερή για να χρησιμοποιηθεί από τον επεξεργαστή. Αυτό μπορεί να παρακολουθηθεί διαβάζοντας το LOCK bit του PLLSTATUS καταχωρητή. Μόλις το LOCK bit γίνει ‘1’, το PLL μπορεί να χρησιμοποιηθεί ως η κύρια πηγή ρολογιού του συστήματος. Αυτό γίνεται με το PLLC bit του PLLCON καταχωρητή.

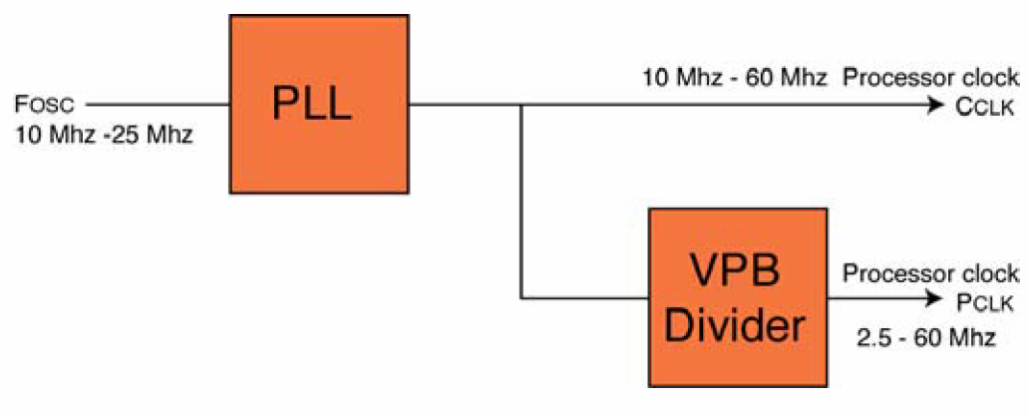


Σχήμα 4-27: Διαδικασία σύνδεσης του PLL.

Πρέπει να προσέξουμε ότι στον καταχωρητή PLLCFG γράφουμε τις τιμές P-1 και M-1. Όταν ο μικροελεγκτής μπαίνει σε κατάσταση μειωμένης κατανάλωσης power-down, το PLL σβύνει και αποσυνδέεται. Όταν ξυπνήσει ο μικροελεγκτής το PLL δεν επανέρχεται στην προηγούμενή του κατάσταση και έτσι πρέπει να επαναπρογραμματίζεται κάθε φορά που το ολοκληρωμένο βγαίνει από κατάσταση μειωμένης κατανάλωσης ενέργειας.

4.15 Διαιρέτης συχνότητας διαδρόμου περιφερειακών

Ο εξωτερικός ταλαντωτής ή η έξοδος του PLL χρησιμοποιείται ως το Cclk που είναι το ρολόι του επεξεργαστή και του AHB διαδρόμου. Τα περιφερειακά είναι συνδεδεμένα σε ξεχωριστό διάδρομο, τον VPB.



Σχήμα 4-28: Ο διαιρέτης συχνότητας λειτουργίας του διαδρόμου περιφερειακών.

Το ρολόι του VPB ονομάζεται Pclk και προκύπτει από το Cclk μέσω της VPB γέφυρας. Η VPB γέφυρα περιέχει έναν διαιρέτη που μπορεί να διαιρέσει το Cclk με την τιμή 1, 2 ή 4. Ο VPB διαιρέτης μπορεί να προγραμματιστεί από τον κώδικα εφαρμογής οποιαδήποτε στιγμή. Κατά την επανεκκίνηση η διαίρεση γίνεται με την μέγιστη τιμή 4, έτσι το Pclk τρέχει με υποτετραπλάσια ταχύτητα από το Cclk. Όλα τα περιφερειακά των LPC2000 μικροελεγκτών μπορούν να τρέξουν με την μέγιστη ταχύτητα των 60MHz ωστόσο όποτε είναι δυνατό από την εφαρμογή είναι καλό να μειώνουμε την ταχύτητα αυτή για εξοικονόμηση ενέργειας.

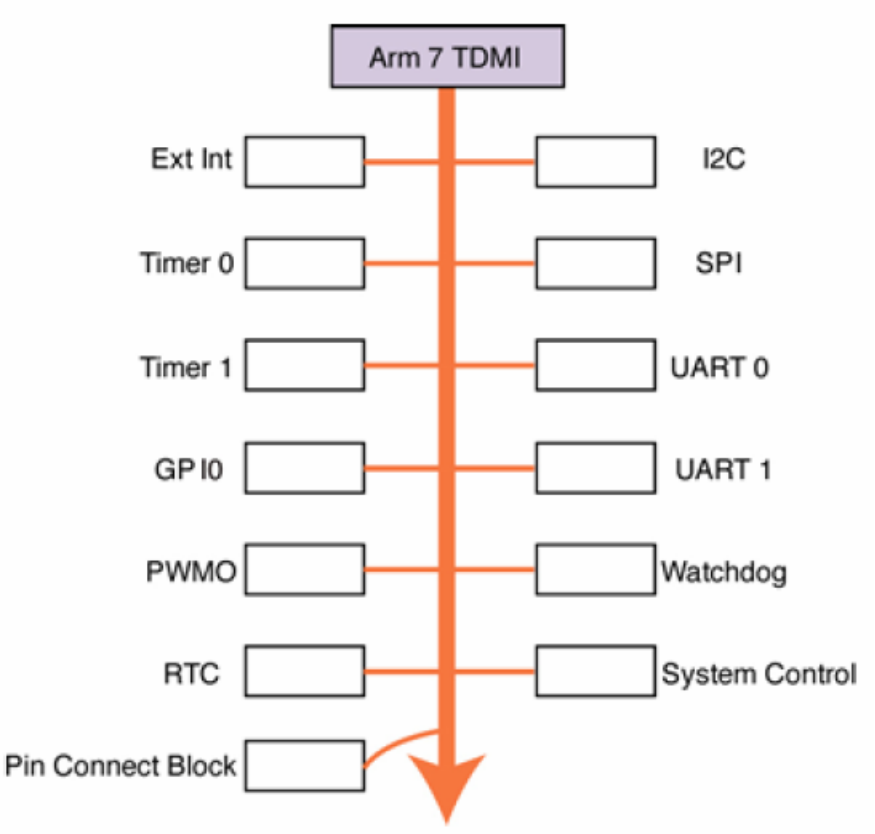
4.16 Έλεγχος κατανάλωσης ισχύος

Η κατανάλωση ισχύος στου μικροελεγκτές είναι ένα μέγεθος που εξαρτάται άμεσα από τον αριθμό των λογικών πυλών και την ταχύτητα λειτουργίας. Αυτό ισχύει και για τους LPC2000 μικροελεγκτές. Η καλή σχεδίαση των LPC2000 συμβάλει στην μικρή κατανάλωση ισχύος. Επίσης η έξυπνη χρήση του PLL και του VPB διαιρέτη μειώνει την ταχύτητα λειτουργίας ολόκληρου του μικροελεγκτή ή των περιφερειακών με αποτέλεσμα την εξοικονόμηση ενέργειας. Ωστόσο οι LPC2000 έχουν επιπλέον συστήματα για την μείωση στην κατανάλωση ενέργειας. Ο ARM7 επεξεργαστής έχει δύο καταστάσεις μειωμένης κατανάλωσης ενέργειας (power down modes) που ελέγχονται από τα δύο πρώτα bits του PCON καταχωρητή. Ο επεξεργαστής μπορεί να μπει σε αδρανή κατάσταση (idle mode) στην οποία ο επεξεργαστής σταματάει αλλά τα περιφερειακά συνεχίζουν να λειτουργούν. Οποιαδήποτε διακοπή από τα περιφερειακά θα ξυπνήσει τον επεξεργαστή και η λειτουργία θα συνεχιστεί.

Ο ARM7 μπορεί να τεθεί επίσης σε κατάσταση χαμηλής ενέργειας (power down mode) στην οποία και ο επεξεργαστής και τα περιφερειακά σταματούν την λειτουργία τους καθώς και το ρολόι του συστήματος. Σε αυτή την κατάσταση μόνο μια επανεκκίνηση ή μια εξωτερική διακοπή από τα pins του μικροελεγκτή θα τον αναγκάσουν να ξυπνήσει. Όλες οι εσωτερικές καταστάσεις των καταχωρητών του επεξεργαστή και η εσωτερική SRAM διατηρούνται, όπως επίσης και οι στατικές λογικές τιμές των ακίδων

εισόδου/έξόδου του μικροελεγκτή. Όταν ξυπνήσει ο μικροελεγκτής ο εξωτερικός ταλαντωτής είναι η πηγή του ρολογιού του συστήματος και συνεπώς το PLL πρέπει να επαναρυθμιστεί.

Το LPC2000 έχει έναν εσωτερικό χρονομετρητή αφύπνησης που εξασφαλίζει ότι το εξωτερικό ρολόι είναι σταθερό και η εσωτερική μνήμη και τα περιφερειακά έχουν αρχικοποιηθεί πριν ο επεξεργαστής αρχίσει να εκτελεί εντολές. Όταν ξυπνήσει ο μικροελεγκτής, ο ταλαντωτής θα αρχίσει να συντονίζεται. Όταν οι παλμοί που παράγει γίνουν αρκετά δυνατοί για να οδηγήσουν το ολοκληρωμένο, ο χρονομετρητής αφύπνισης θα μετρήσει 4096 παλμούς πριν αρχικοποιήσει την μνήμη Flash και ξαναρχίσει την εκτέλεση του προγράμματος. Αυτό εξασφαλίζει την μικρότερη καθυστέρηση επανεκκίνησης μετά από την κατάσταση χαμηλής ενέργειας ή επανεκκίνησης του ολοκληρωμένου. Είναι επίσης δυνατό η απενεργοποίηση μεμονωμένων περιφερειακών αν δεν χρησιμοποιούνται, από το bit ελέγχου ενέργειας (power control bit) στον PCONP καταχωρητή. Μερικά περιφερειακά δεν μπορούν να απενεργοποιηθούν αυτά είναι : ο χρονομετρητής επιτήρησης (watchdog timer), οι γενικές χρήσεις είσοδοι/έξοδοι (GPIO), pin connect block και system control block.

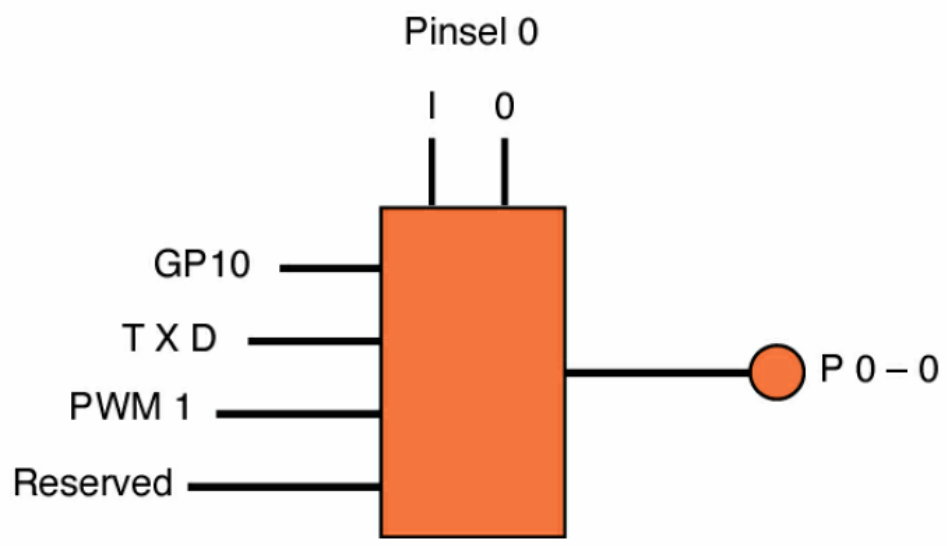


Σχήμα 4-29: Τα περιφερειακά του LPC2148 είναι ανεξάρτητα μεταξύ τους

4.17 Το σύστημα διακοπών του LPC2000

4.17.1 Σύστημα επιλογής λειτουργίας των πινς

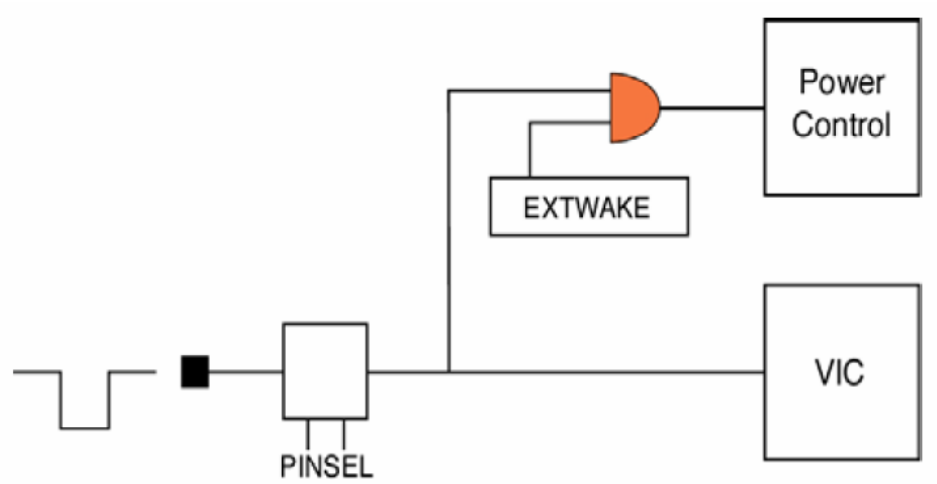
Όλα τα pins εισόδου/εξόδου των μικροελεγκτών LPC2000 είναι συνδεδεμένα με διάφορες εσωτερικές λειτουργίες μέσω ενός πολυπλέκτη που ονομάζεται σύστημα επιλογής πιν (pin select block). Αυτός ο πολυπλέκτης επιτρέπει στον χρήστη να ρυθμίσει κάποιο pin ώς γενική είσοδο/έξοδο ή να επιλέξει κάποια από τρεις άλλες λειτουργίες. Κατά την επανεκκίνηση όλα τα pins εισόδου/εξόδου είναι ρυθμισμένα ως γενικές είσοδοι/έξοδοι GPIO. Οι δευτερεύουσες λειτουργίες επιλέγονται μέσω των PINSEL καταχωρητών.



Σχήμα 4-30: Το σύστημα επιλογής λειτουργίας των πινς.

4.17.2 Τα πινς των εξωτερικών διακοπών

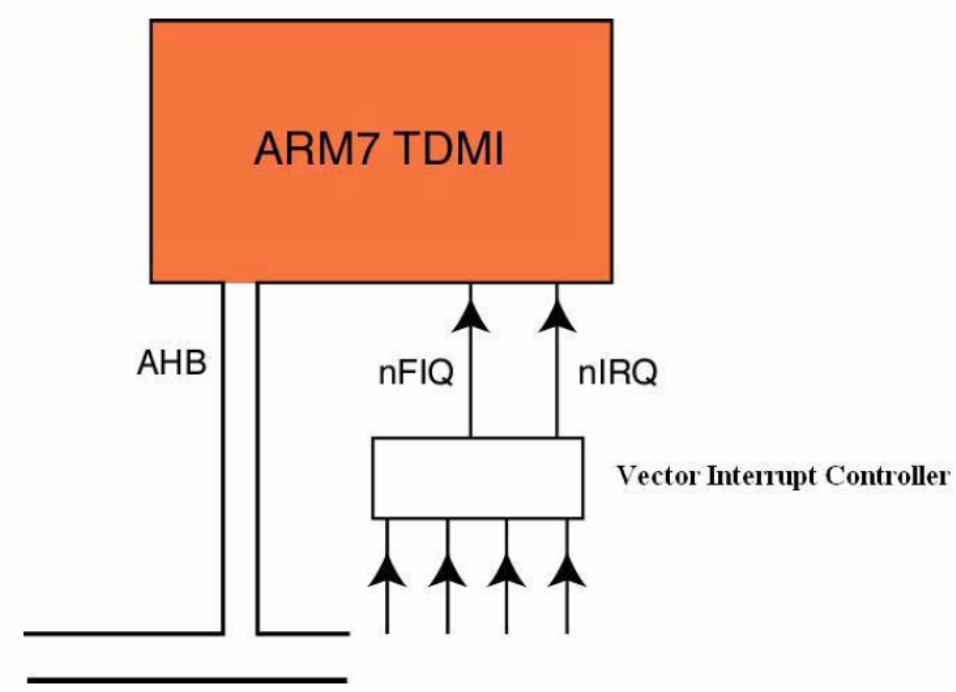
Οι εξωτερικές διακοπές ελέγχονται από τέσσερις καταχωρητές. Ο EXTMODE καταχωρητής ελέγχει αν η διακοπή θα είναι ευαίσθητη σε λογική στάθμη (level sensitive) ή σε αλλάγη λογικής στάθμης (edge sensitive). Αν η εξωτερική διακοπή ρυθμιστεί να είναι ευαίσθητη σε αλλαγή λογικής στάθμης, ο EXPOL καταχωρητής χρησιμοποιείται για να καθορίσει αν θα γίνεται κατά την θετική ή την αρνητική ακμή του παλμού. Οι εξωτερικές διακοπές που είναι ευαίσθητες σε λογική στάθμη μπορούν να προκληθούν μόνο από μηδενική λογική στάθμη. Αν χρησιμοποιείται κατάσταση χαμηλής κατανάλωσης, ο EXTWAKE καταχωρητής μπορεί να ρυθμιστεί ώστε να ξυπνάει ο επεξεργαστής από μια διακοπή.



Σχήμα 4-31: Οι εξωτερικές διακοπές μπορούν να ξυπνήσουν τον επεξεργαστή από κατάσταση μειωμένης κατανάλωσης ενέργειας.

4.17.3 Δομή διακοπών

Ο ARM7 επεξεργαστής έχει δύο γραμμές εξωτερικών διακοπών, μία για την γρήγορη διακοπή (Fast interrupt request) FIQ και μία για τις γενικές εξωτερικές διακοπές (General purpose interrupt request) IRQ. Σαν γενικός κανόνας, οι γρήγορες διακοπές πρέπει να εξυπηρετούν μόνο μια πηγή διακοπών ώστε μόλις συμβεί διακοπή, να εκτελείται αμέσως η ρουτίνα εξυπηρέτησης διακοπής ώστε να είναι όσο τον δυνατόν πιο γρήγορη η εξυπηρέτησή της. Αυτό σημαίνει ότι όλες οι άλλες πηγές διακοπών πρέπει να εξυπηρετούνται από IRQ διακοπές. Σε ένα απλό σύστημα οι πηγές αυτές θα μπορούσαν να είναι συνδεδεμένες σε μια μεγάλη OR-πύλη. Με αυτόν τον τρόπο όταν προκληθεί διακοπή, ο επεξεργαστής πρέπει να ελέγξει κάθε περιφερειακό ώστε να βρει την πηγή της διακοπής. Αυτό μπορεί να πάρει πολλούς κύκλους. Ο ARM7 για να λύσει αυτό το πρόβλημα αποδοτικά έχει μια εσωτερική μονάδα που ονομάζεται ελεγκτής διανυσμάτων διακοπών (Vector Interrupt Controller) VIC.



Σχήμα 4-32: Μπλοκ διάγραμμα της λειτουργίας του VIC.

Ο VIC χρησιμοποιείται για να χειρίζεται όλες τις διακοπές από τα περιφερειακά του μικροελεγκτή. Κάθε πηγή διακοπής είναι συνδεδεμένη στον VIC σε καθορισμένο κανάλι. Ο κώδικας εφαρμογής μπορεί να συνδέσει κάθε πηγή διακοπής στον επεξεργαστή με τρεις διαφορετικούς τρόπους. Ο VIC επιτρέπει σε κάθε διακοπή να χειριστεί σαν FIQ διακοπή, διανυσματική διακοπή IRQ (vectored interrupt) ή μη διανυσματική διακοπή IRQ (non vectored interrupt). Ο χρόνος ανταπόκρισης σε κάθε τύπου διακοπή είναι διαφορετικός. Η FIQ είναι η πιο γρήγορη, μετά ακολουθεί η διανυσματική διακοπή και τέλος η μη διανυσματική διακοπή που είναι η πιο αργή.

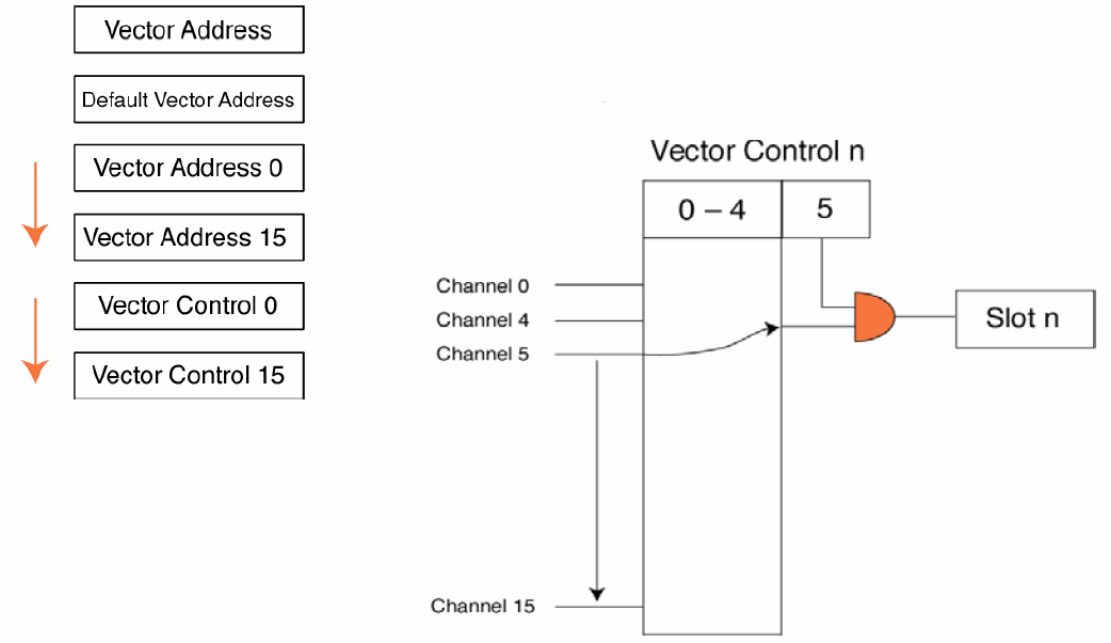
4.17.4 Γρήγορες διακοπές (FIQ)

Καθε πηγή διακοπών μπορεί να καθορίστει ώς γρήγορη διακοπή FIQ. Ο καταχωρητής επιλογής διακοπών VIC (VIC interrupt select register) VICIntSelect έχει ένα ξεχωριστό bit για κάθε διακοπή. Κάνοντας ‘1’ αυτό το bit ρυθμίζει το κανάλι επιλογής ως FIQ διακοπή. Ένα ιδανικό σύστημα έχει μόνο μία γρήγορη διακοπή. Ωστόσο θέτοντας πολλά bits στον VICIntSelect, καθορίζονται πολλές πηγές για τη FIQ διακοπή. Έτσι ο κώδικας πρέπει να ελέγχει τον καταχωρητή FIQ κατάστασης (VIC FIQ Status register) VICFIQStatus για το ποια πηγή δημιούργησε την διακοπή και να εκτελείται το αντίστοιχο τμήμα του. Έχοντας όμως καθορίσει πολλές πηγές για FIQ διακοπή, η εκτέλεσή της γίνεται πιο αργή που είναι αντίθετη στην λειτουργία της. Όταν καθοριστεί η πηγή της FIQ διακοπής, η διακοπή μπορεί να ενεργοποιηθεί με τον (VIC Interrupt Enable register). Επίσης η διακοπή πρέπει να ενεργοποιηθεί και από το περιφερειακό που θα την δημιουργεί. Μόλις η FIQ διακοπή δημιουργηθεί, ο επεξεργαστής θα μπει σε κατάσταση FIQ και η επόμενη εντολή που θα εκτελέσει βρίσκεται στην FIQ διεύθυνση 0x0000001C. Εκεί ο χρήστης πρέπει να έχει τοποθετήσει μια εντολή άλματος που θα οδηγεί στην ρουτίνα εξυπηρέτησης διακοπής. Πριν γίνει έξοδος από

την ρουτίνα εξυπηρέτησης διακοπής, ο χρήστης πρέπει να ελέγχει αν έχει μηδενιστεί το bit της σημαίας κατάστασης της διακοπής (interrupt status flag) του περιφερειακού. Αν δεν έχει μηδενιστεί πρέπει να γράψουμε '1' στην σημαία ώστε να ενημερωθεί και να μηδενιστεί αλλιώς θα γίνεται συνέχεια η ίδια διακοπή.

4.17.5 Διανυσματικές διακοπές (Vectored IRQ)

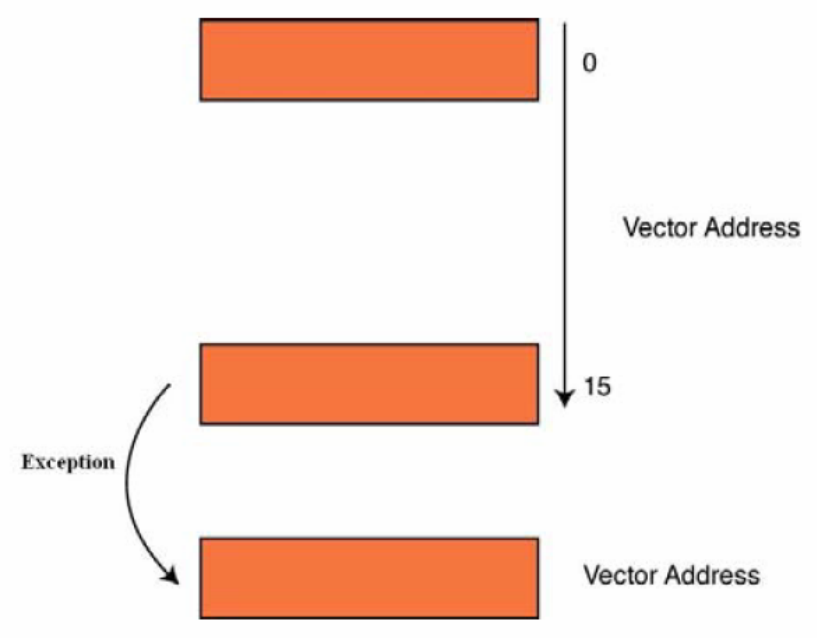
Αν έχουμε καθορίσει μια πηγή διακοπής ως FIQ όλες οι υπόλοιπες πηγές διακοπών πρέπει να καθοριστούν ως διανυσματικές διακοπές IRQ. Για την καλύτερη και γρήγορη εξυπηρέτηση των διακοπών, ο VIC διαθέτει έναν πίνακα 16 θέσεων που η κάθε θέση δείχνει την διεύθυνση μιας ρουτίνας εξυπηρέτησης διακοπής μέσω του καταχωρητή διευθύνσεων διανυσμάτων (Vector Address register) και την προτεραιότητά της μέσω του καταχωρητή ελέγχου διανυσμάτων (Vector Control register). Ο πίνακας αυτός μπορεί να προγραμματιστεί από τον χρήστη.



Σχήμα 4-33: Σύστημα διανυσματικών διακοπών.

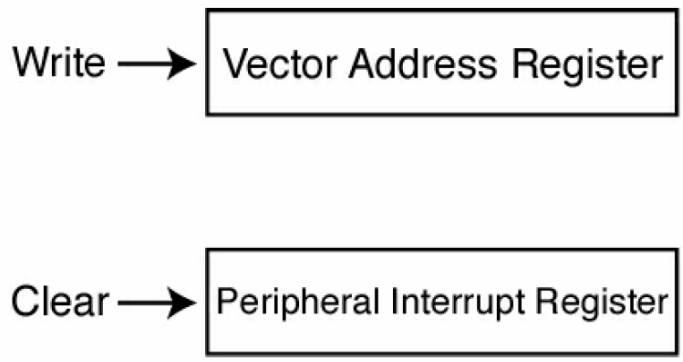
Ο καταχωρητής ελέγχου διανυσμάτων περιέχει δύο πεδία: ένα πεδίο καναλιών και ένα bit ενεργοποίησης. Προγραμματίζοντας το πεδίο καναλιού, κάθε κανάλι διακοπών μπορεί να συνδεθεί σε κάποια σχισμή του VIC και μετά να ενεργοποιηθεί μέσω του bit ενεργοποίησης. Η προτεραιότητα της κάθε διακοπής καθορίζεται από τον αριθμό της σχισμής, όσο μικρότερη είναι η τιμής της σχισμής τόσο πιο σημαντική είναι η διακοπή.

Ο καταχωρητής διευθύνσεων διανυσμάτων πρέπει να αρχικοποιηθεί με τις διευθύνσεις των C συναρτήσεων που πρέπει να τρέχουν όταν ενεργοποιείται κάποια διακοπή που σχετίζεται με τις σχισμές.



Σχήμα 4-34: Οι καταχωρητές διευθύνσεων διανυσμάτων.

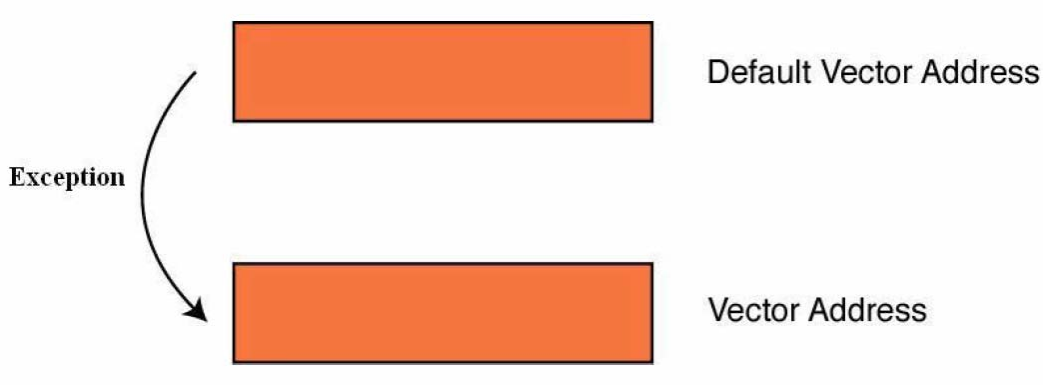
Όταν συμβεί μια διακοπή ενημερώνεται ο μετρητής προγράμματος PC με την διεύθυνση της ρουτίνας που δείχνει ο καταχωρητής διευθύνσεων διανυσμάτων και συνεπώς γίνεται η εκτέλεση της ρουτίνας. Όταν τελειώσει η ρουτίνα εξυπηρέτησης διακοπής πρέπει να μηδενίσουμε την σημαία κατάστασης διακοπής του περιφερειακού για να μην επαναληφθεί η διακοπή. Επίσης πρέπει να γράψουμε στον καταχωρητή διευθύνσεων διανυσμάτων μια τιμή ώστε να ενημερωθεί ο VIC και να καταλάβει ότι τελείωσε η εξυπηρέτηση διακοπής ώστε να μπορούν να εκτέλεστούν και άλλες διακοπές αν χρειαστεί.



Σχήμα 4-35: Διαδικασία εξόδου από διανυσματική διακοπή.

4.17.6 Μη διανυσματικές διακοπές

Ο VIC μπορεί να χειριστεί 16 διανυσματικές διακοπές IRQ και τουλάχιστον μία γρήγορη FIQ διακοπή. Αν υπάρχουν περισσότερες από 17 πηγές διακοπών στον μικροελεγκτή πρέπει να εξυπηρετηθούν ως μη διανυσματικές. Οι μη διανυσματικές διακοπές εξυπηρετούνται από μία μόνο ρουτίνα εξυπηρέτησης διακοπής (ISR). Η διεύθυνση αυτής της ρουτίνας είναι αποθηκευμένη στον καταχωρητή προεπιλεγμένης διεύθυνσης διανύσματος (default vector address register). Αν έχει ενεργοποιηθεί μια διακοπή και δεν έχει καθοριστεί ως FIQ ή διανυσματική IRQ διακοπή τότε θα εξυπηρετηθεί ώς μη διανυσματική. Όταν πάει να εκτελεστεί μια τέτοια διακοπή, η διεύθυνση που βρίσκεται στον προεπιλεγμένο καταχωρητή θα μεταφερθεί στον καταχωρητή διεύθυνσης διανύσματος προκαλώντας τον επεξεργαστή να εκτελέσει την ρουτίνα που δείχνει η διεύθυνση αυτή. Στην αρχή της ρουτίνας πρέπει να ελέγχεται ο καταχωρητής κατάστασης IRQ (IRQ Status register) για να εξακριβωθεί ποια πηγή διακοπών προξένησε τη διακοπή ώστε να τρέξει το αντίστοιχο κομμάτι κώδικα. Στην έξοδο από τη μη διανυσματική διακοπή πρέπει να καθαρίσουμε τη σημαία κατάστασης διακοπής του περιφερειακού και να γράψουμε μια τιμή στο καταχωρητή διεύθυνσης διανύσματος για να ενημερωθεί ο VIC.

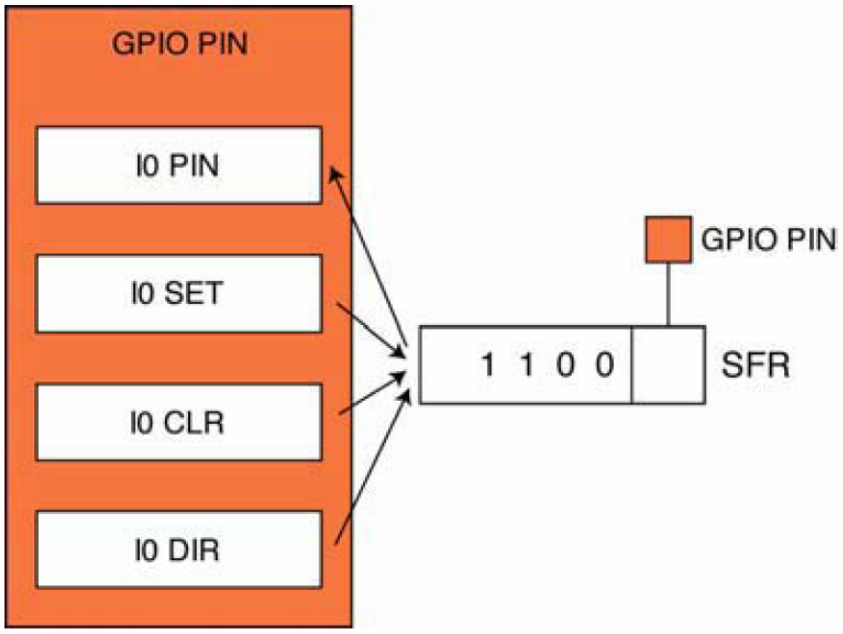


Σχήμα 4-36: Στις μη διανυσματικές διακοπές υπάρχει μια διεύθυνση ρουτίνας εξυπηρέτησης διακοπής.

Επιπλέον ο VIC έχει μια προστατευμένη κατάσταση που δεν επιτρέπει πρόσβαση σε κανέναν από τους καταχωρητές του σε κατάσταση χρήστη (User mode). Αν ο κώδικας εφαρμογής επιθυμεί να προσπελάσει τον VIC, αυτό πρέπει να γίνει σε προνομιακή κατάσταση. Αυτό γίνεται μέσα σε μια IRQ, FIQ διακοπή ή τρέχοντας μια SWI εντολή.

4.18 Είσοδοι/έξοδοι γενικού σκοπού

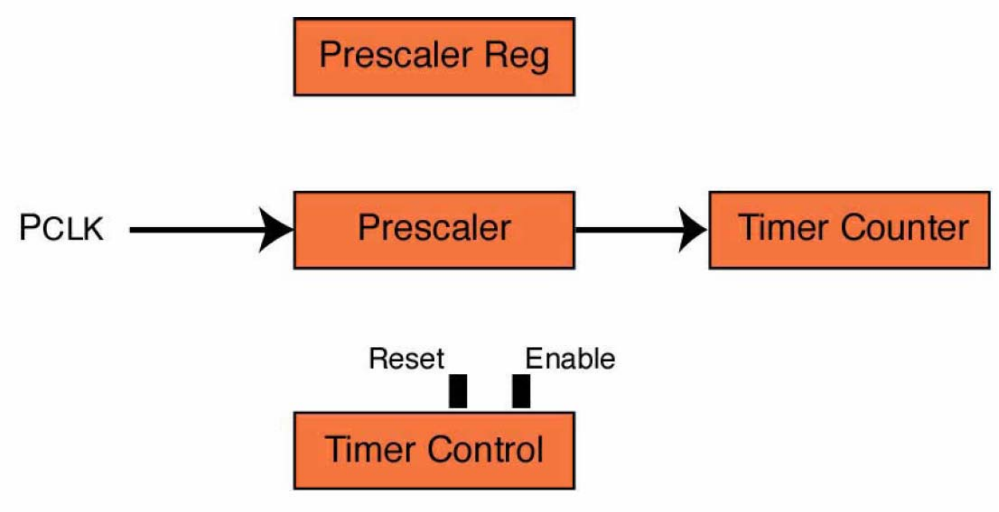
Κατά την επανεκκίνηση του μικροελεγκτή όλα τα περιφερειακά πινς είναι ρυθμισμένα σε λειτουργία εισόδου/εξόδου γενικού σκοπού. Τα πινς εισόδου/εξόδου ελέγχονται από τέσσερις καταχωρητές. Ο IODIR καταχωρητής καθορίζει μέσω των bit του αν τα πινς που τους αντιστοιχούν είναι είσοδοι (‘0’) ή έξοδοι (‘1’). Αν κάποιο πιν καθοριστεί σαν έξοδος τότε οι καταχωρητές IOSET και IOCLR καθορίζουν την λογική τιμή εξόδου αυτού του πιν. Γράφοντας ‘1’ σε bit αυτών των καταχωρητών γίνονται ‘1’ ή ‘0’ αντίστοιχα οι έξοδοι. Η κατάσταση των πινς εισόδου/εξόδου μπορεί να διαβαστεί οποιαδήποτε στιγμή από το περιεχόμενο του IOPIN καταχωρητή.



Σχήμα 4-37: Καταχωρητές ελέγχου εισόδου/εξόδου γενικού σκοπού.

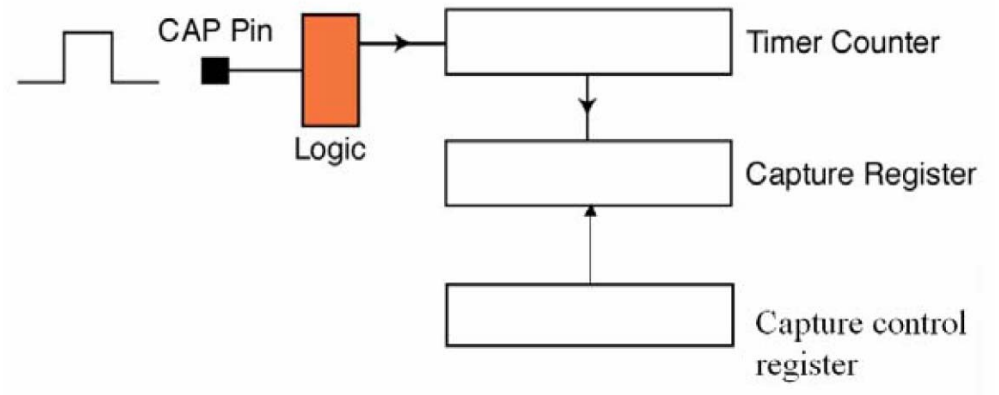
4.19 Χρονομετρητές γενικού σκοπού

Ο LPC2148 έχει δύο γενικού σκοπού χρονομετρητές. Οι χρονομετρητές βασίζονται σε 32-bit μετρητές χρόνου (timer counters) με 32-bit καταχωρητές υποδιαίρεσης (prescalers). Η πηγή παλμών για όλους τους μετρητές είναι το ρολόι VLSI περιφερειακών Pclk.



Σχήμα 4-38: Χρονομετρητής γενικού σκοπού.

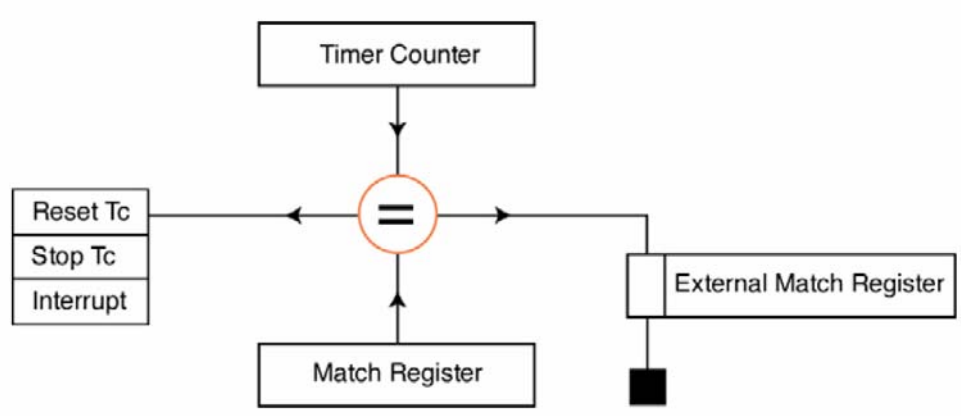
Ο ρυθμός λειτουργίας του χρονομετρητή ελέγχεται από την τιμή που αποθηκεύεται στον καταχωρητή υποδιαίρεσης. Ο μετρητής υποδιαίρεσης αυξάνει την τιμή του κατά ένα με κάθε παλμό του Pclk μέχρι να φτάσει στην τιμή που έχει ο καταχωρητής υποδιαίρεσης. Όταν γίνει αυτό η τιμή του χρονομετρητή αυξάνεται κατά ένα ενώ ο μετρητής υποδιαίρεσης μηδενίζεται και συνεχίζει ξανά από την αρχή την λειτουργία του. Ο καταχωρητής ελέγχου χρονομετρητή (Timer control register) περιέχει μόνο δύο bits που χρησιμεύουν για να ενεργοποιήσουν/απενεργοποιήσουν τον χρονομετρητή και να τον επανεκκινήσουν (μηδενίσουν).



Σχήμα 4-39: Λειτουργία σύλληψης του χρονομετρητή.

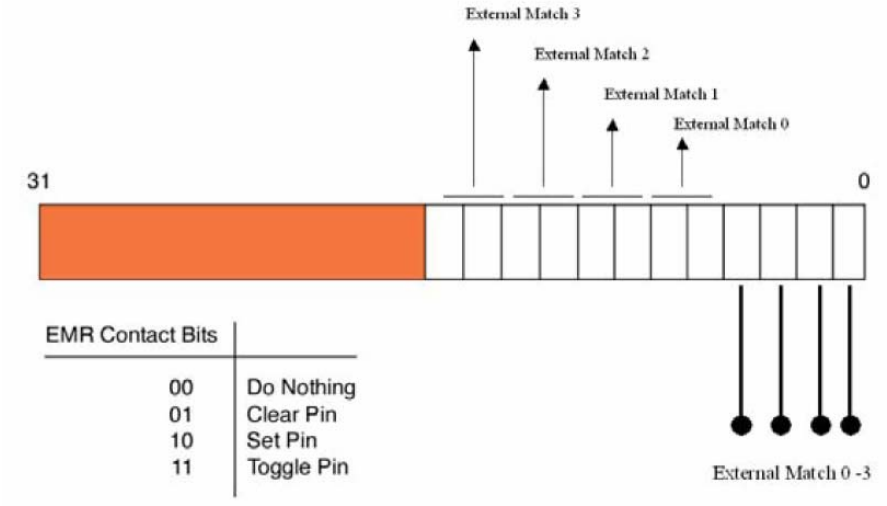
Επιπλέον εκτός από την λειτουργία μέτρησης χρόνου, κάθε χρονομετρητής έχει και τέσσερα κανάλια σύλληψης (capture channels). Τα κανάλια σύλληψης επιτρέπουν την καταγραφή της τιμής του χρονομετρητή όταν ένα σήμα εισόδου κάνει μια αλλαγή στην τιμή του. Κάθε κανάλι σύλληψης έχει ένα πιν σύλληψης που παίρνει το σήμα εισόδου και μπορεί να ενεργοποιηθεί μέσω του pin connect block. Ο καταχωρητής ελέγχου σύλληψης (Capture Control register) μπορεί να καθορίσει αν μια θετική, αρνητική ή και οι δύο ακμές σε αυτό το πιν, μπορούν να δημιουργήσουν ένα συμβάν σύλληψης. Όταν το συμβάν σύλληψης δημιουργηθεί, η τρέχουσα τιμή του χρονομετρητή θα μεταφερθεί στο κατάλληλο καταχωρητή σύλληψης και αν έχει ρυθμιστεί από τον κώδικα εφαρμογής, αυτό μπορεί να δημιουργησεί διακοπή.

Κάθε χρονομετρητής έχει μέχρι τέσσερα κανάλια ισοτιμίας (match channels). Κάθε τέτοιο κανάλι έχει έναν καταχωρητή ισοτιμίας (match register) που αποθηκεύει έναν αριθμό των 32-bit. Η τρέχουσα τιμή του χρονομετρητή συγκρίνεται συνεχώς με τον καταχωρητή ισοτιμίας. Όταν οι τιμές βρεθούν ίδιες, ένα συμβάν ισοτιμίας δημιουργείται. Αυτό το συμβάν μπορεί να προκαλέσει μια αλλαγή στον χρονομετρητή (μηδενισμό, σταμάτημα), να δημιουργήσει μια διακοπή ή να επηρεάσει ένα εξωτερικό πιν όπως να το κάνει λογικό ΄1΄, ‘0’ ή να μεταβάλλει την κατάστασή του.

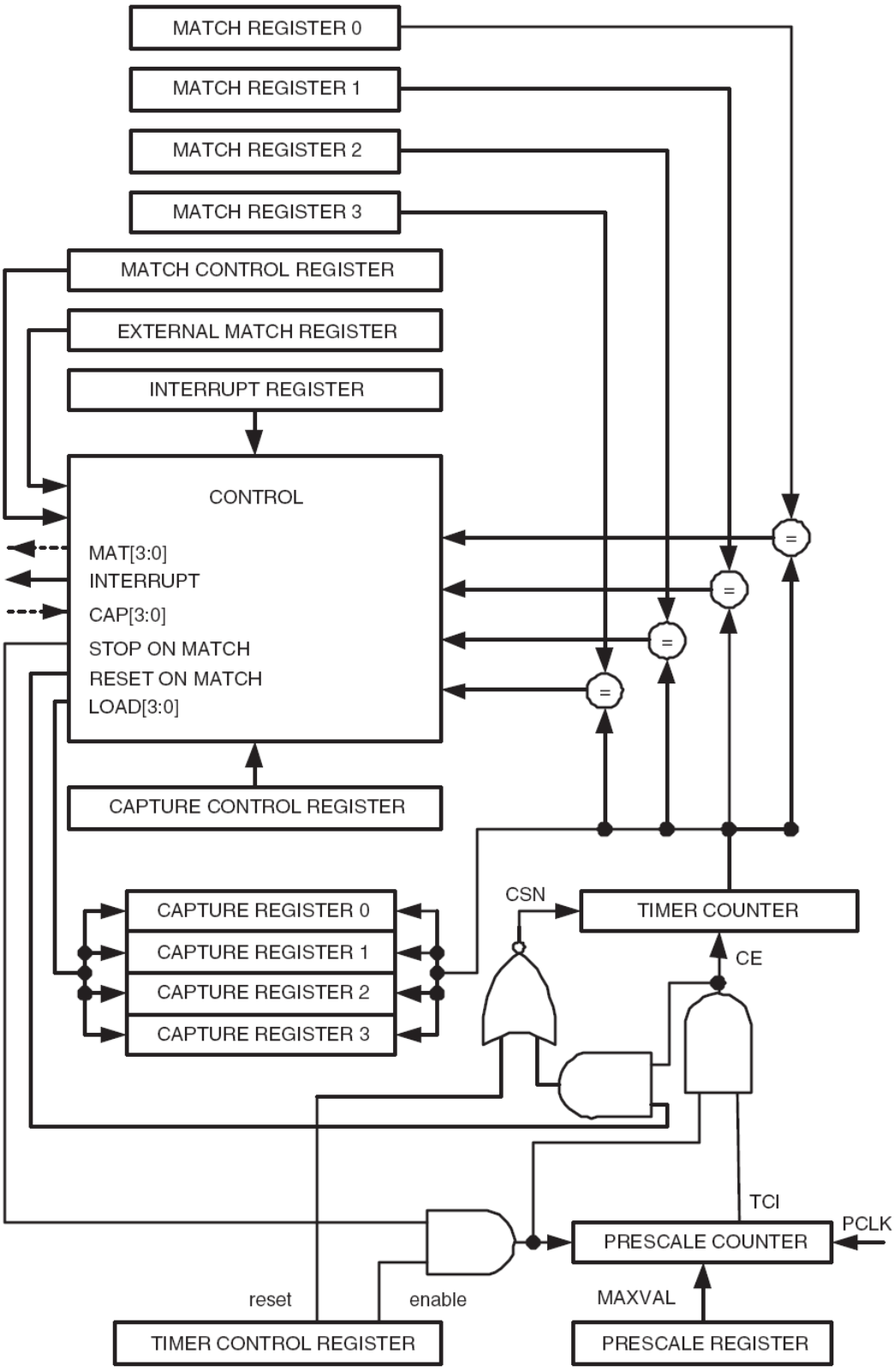


Σχήμα 4-40: Λειτουργία ισοτιμίας του χρονομετρητή.

Για να ρυθμίσουμε τον χρονομετρητή σε λειτουργία για συμβάν ισοτιμίας, πρέπει να φορτώσουμε τον μετρητή ισοτιμίας με την επιθυμητή τιμή. Το εσωτερικό συμβάν ισοτιμίας μπορεί να ρυθμιστεί με τον καταχωρητή ελέγχου ισοτιμίας (match control register). Σε αυτόν τον καταχωρητή κάθε κανάλι έχει μια ομάδα από bits που μπορούν να χρησιμοποιηθούν για να ενεργοποιήσουν τις ακόλουθες ενέργειες σε ένα συμβάν ισοτιμίας: δημιουργία διακοπής (interrupt) χρονομετρητή, μηδενισμός ή σταμάτημα του χρονομετρητή. Οποιοσδήποτε συνδυασμός από αυτές τις ενέργειες μπορεί να πραγματοποιηθεί. Επιπλέον κάθε κανάλι ισοτιμίας έχει ένα αντίστοιχο πιν που μπορεί να αλλάξει τιμή σε κάθε συμβάν αρκεί να έχει ενεργοποιηθεί στο pin connect block.



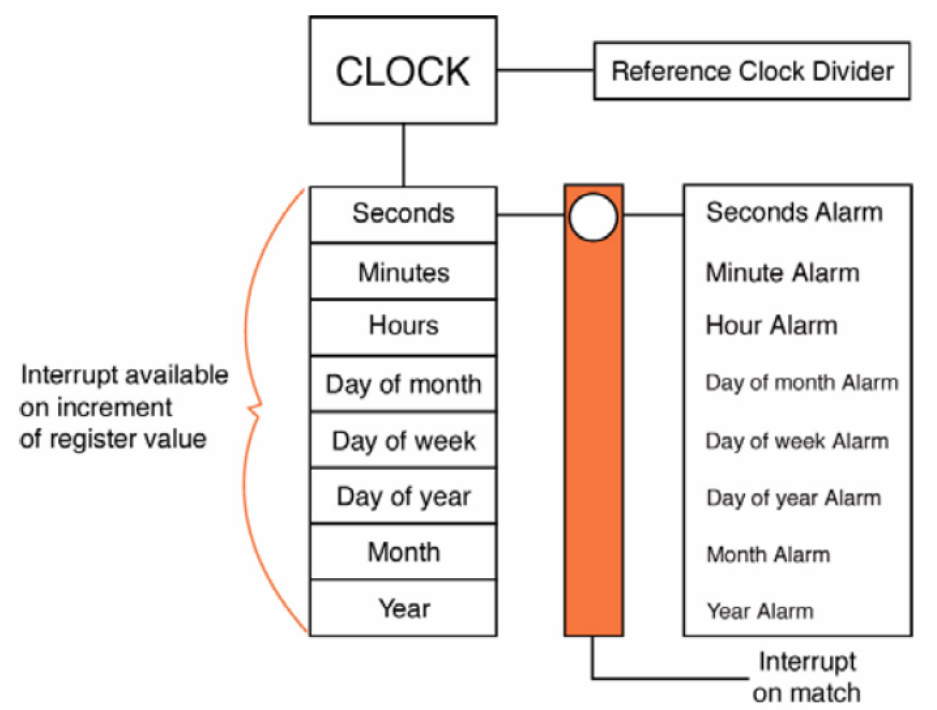
Σχήμα 4-41: Έλεγχος εξόδων σε συμβάν ισοτιμίας.



Σχήμα 4-42: Μπλοκ διάγραμμα χρονομετρητή.

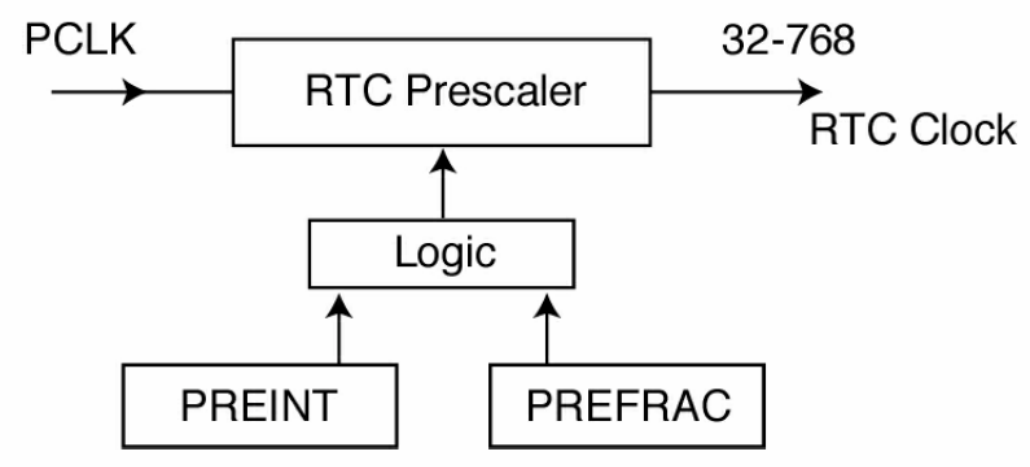
4.20 Ρολόι πραγματικού χρόνου

Οι LPC2xxx μικροελεγκτές έχουν ένα ρολόι πραγματικού χρόνου (Real Time Clock) RTC το οποίο μπορεί να κρατάει με ακρίβεια ημερομηνίες και ώρα έως το έτος 2099. Όπως όλα τα υπόλοιπα περιφερειακά το RTC τρέχει από το Pclk έτσι δεν χρειάζεται επιπρόσθετος εξωτερικός ταλαντωτής. Έχει σχεδιαστεί να καταναλώνει μικρή ενέργεια και είναι ιδανικό για εφαρμογές που χρησιμοποιούν μπαταρίες. Εκτός από την λειτουργία του να κρατάει την ημερομηνία, το RTC έχει ειδικούς καταχωρητές συναγερμού (alarm registers) οι οποίοι μπορούν να ρυθμιστούν και να παράγουν κάποιο συμβάν σε συγκεκριμένες ημερομηνίες, ώρες ή με κάποια τιμή που κρατείται σε έναν ειδικό καταχωρητή.



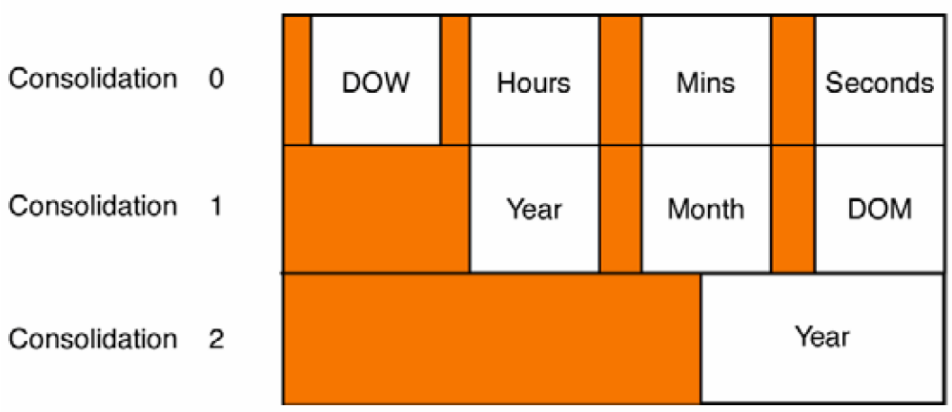
Σχήμα 4-43: Λειτουργία ρολογιού πραγματικού χρόνου.

Το RTC τρέχει από την συχνότητα των 32.7kHz. Για να δημιουργήσει αυτή τη συχνότητα το Pclk είναι συνδεδεμένο σε έναν διαιρέτη ο οποίος μπορεί να διαιρέσει οποιαδήποτε συχνότητα του Pclk. Ο διαιρέτης αυτός ρυθμίζεται από τους καταχωρητές PREINT και PREFRAC οι οποίοι κρατούν την ακέραια και δεκαδική τιμή του διαιρέτη αντίστοιχα.



Σχήμα 4-44: Ρύθμιση συχνότητας ρολογιού πραγματικού χρόνου.

Υπάρχουν οκτώ καταχωρητές που χρησιμοποιούνται για την μέτρηση ημερομηνίας και ώρας. Ο καθένας περιέχει μια μοναδική τιμή που αντιστοιχεί στο μέγεθος που μετράει. Αυτοί είναι: μέρα της εβδομάδας (DOW), ώρες, λεπτά, δευτερόλεπτα τα οποία μπορούν να διαβαστούν με μία ανάγνωση από έναν μεγάλο καταχωρητή, έτος, μήνας, μέρα του μήνα (DOM) που ομοίως μπορούν να διαβαστούν όλα μαζί και τέλος το έτος.

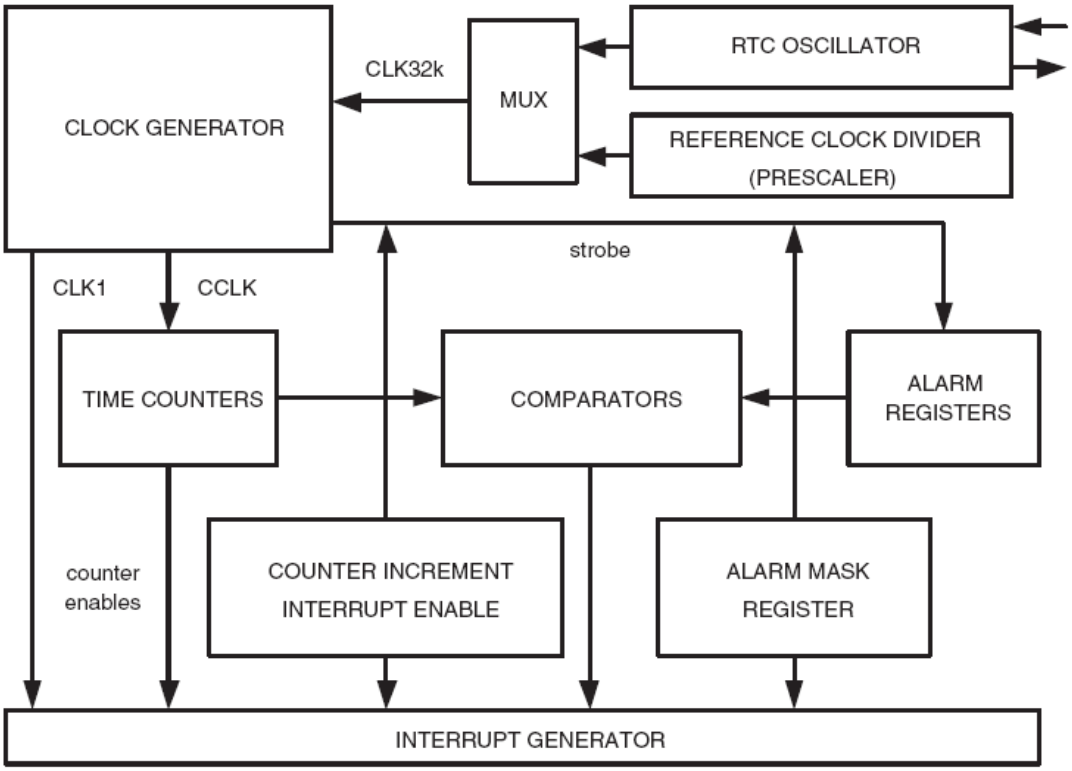


Σχήμα 4-45: Καταχωρητές μέτρησης ημερομηνίας και ώρας του RTC.

Το RTC επιτρέπει την δημιουργία διακοπής (interrupt) κάθε φορά που αυξάνεται κάποιο από τα οκτώ μεγέθη. Έτσι μπορεί να δημιουργείται διακοπή κάθε δευτερόλεπτο ή κάθε φορά που αλλάζει ο μήνας.

Υπάρχει και μία δεύτερη μέθοδος δημιουργίας διακοπής η οποία χρησιμοποιεί τους καταχωρητές συναγερμού. Καθένας από τους οκτώ καταχωρητές που μετράνε την ημερομηνία-ώρα έχει έναν δικό του καταχωρητή συναγερμού. Αν ο καταχωρητής συναγερμού είναι επιλεγμένος (μέσω του Alarm mask register), συγκρίνεται με τον αντίστοιχο καταχωρητή που μετράει τον χρόνο και αν οι τιμές τους ταιριάζουν δημιουργείται μια διακοπή. Με αυτό τον τρόπο είναι δυνατό να καθορίσουμε μια διακοπή μέχρι το 2099 με ενός δευτερολέπτου ακρίβεια.

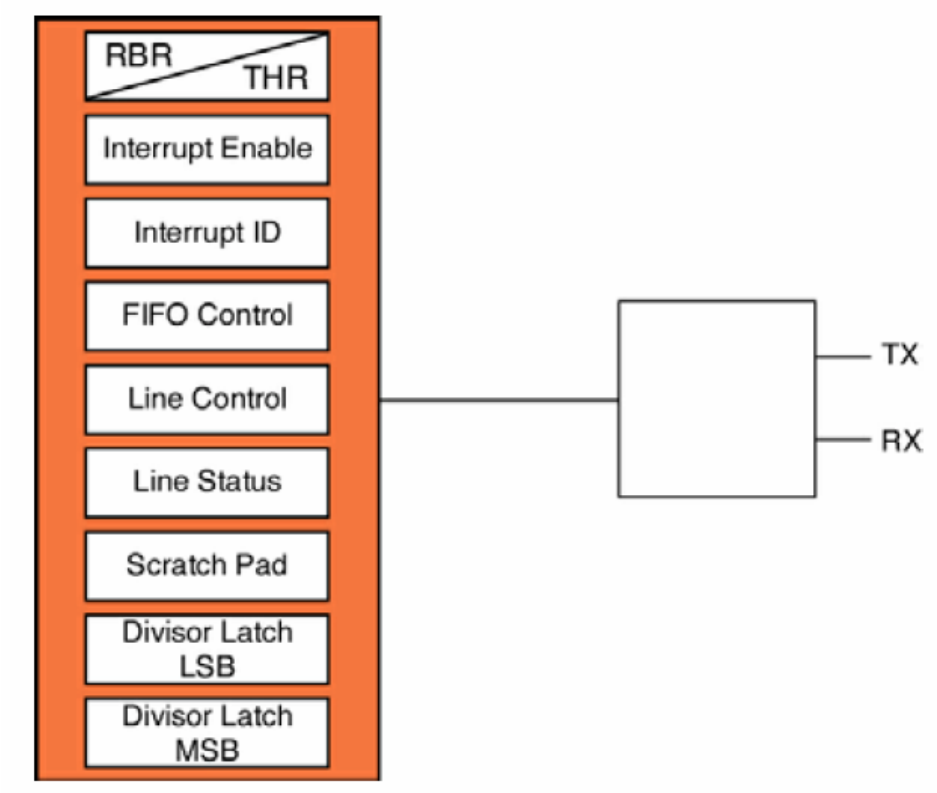
Επειδή υπάρχουν δύο τρόποι που δημιουργούν RTC διακοπή, μέσα στην ρουτίνα διακοπής πρέπει να διαβαστούν δύο bit-σημαίες του καταχωρητή Interrupt Location που δείχνουν με ποιον τρόπο δημιουργήθηκε η διακοπή. Πριν γίνει έξοδος από την διακοπή αυτές οι σημαίες πρέπει να καθαριστούν για να μην ξαναγίνει χωρίς λόγο είσοδος στην διακοπή.



Σχήμα 4-46: Μπλοκ διάγραμμα ρολογιού πραγματικού χρόνου.

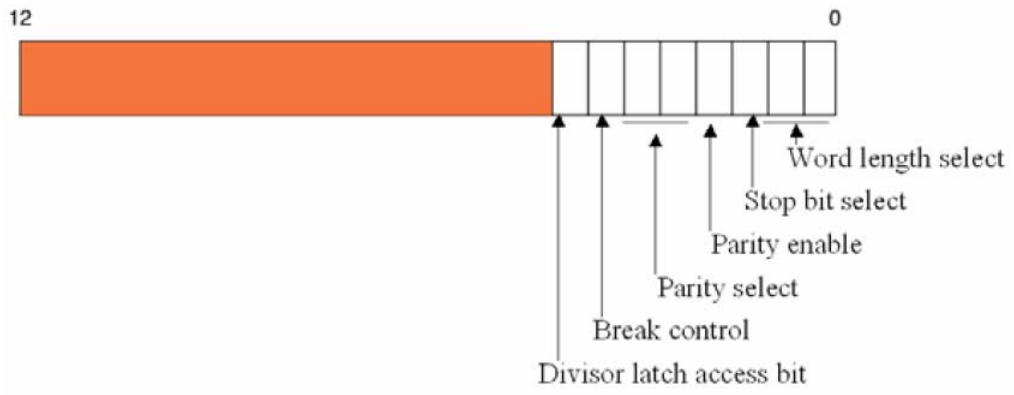
4.21 Περιφερειακό σειριακής μετάδοσης UART

Οι μικροελεγκτές LPC2xxx έχουν δύο εσωτερικά UART's. Χρησιμοποιούνται με τον ίδιο τρόπο με την διαφορα ότι το UART1 έχει και υποστήριξη για modem. Και τα δύο έχουν εσωτερική γεννήτρια ρυθμού μετάδοσης (baud rate) και FIFO's αποστολής και λήψης μεγέθους16 byte.



Σχήμα 4-47: Λειτουργία UART.

Αρχικά πρέπει να ρυθμιστούν τα πινς που αντιστοιχούν στο UART σε Rx και Tx από GPIO. Μετά ο καταχωρητής ελέγχου γραμμής UART (UART Line Control register) UxLCR πρέπει να ρυθμιστεί για να καθοριστεί η μορφή αποστολής του χαρακτήρα μέσω του UART όπως stop-bit, no-parity, 8bits κλπ.

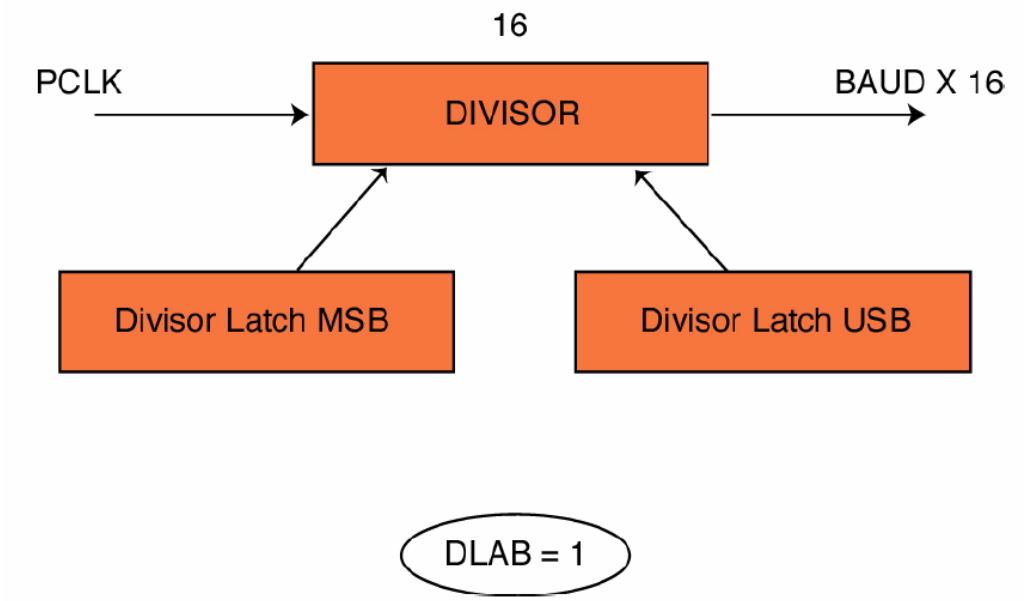


Σχήμα 4-48: Καταχωρητής ελέγχου γραμμής.

Ο ρυθμός μετάδοσης προκύπτει από το Pckl μέσω δύο καταχωρητών διαίρεσης (Divisor Latch registers) UxDLL και UxDLM. Ωστόσο για να προγραμματιστεί ο καταχωρητής αυτός πρέπει το bit7 DLAB του καταχωρητή ελέγχου γραμμής UxLCR να γίνει ‘1’. Για μεγαλύτερη ακρίβεια υπάρχει και ο καταχωρητής κλασματικής διαίρεσης (Fractional Divider register) UxFDR με

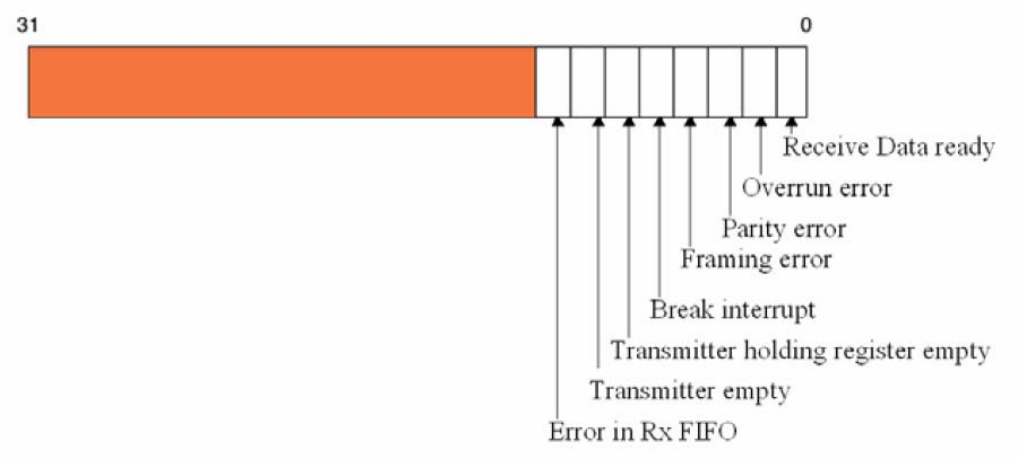
τον οποίο μπορούμε να προγραμματίσουμε τις τιμές DIVADDVAL και MULVAL. Ο ρυθμός μετάδοσης δίνεται από τον τύπο :

$$UART0_{baudrate} = \frac{PCLK}{16 \times (256 \times U0DLM + U0DLL)} \times \frac{MulVal}{(MulVal + DivAddVal)}$$



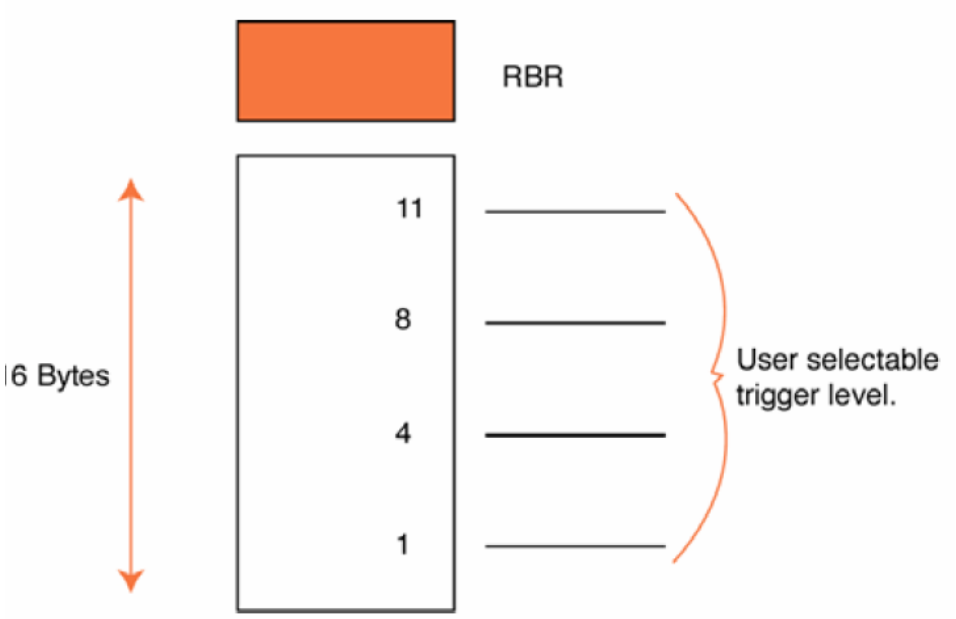
Σχήμα 4-49: Ρύθμιση ταχύτητας μεταφοράς δεδομένων UART.

Όταν αρχικοποιηθεί το UART, μπορούν να σταλθούν χαρακτήρες γράφοντας στον καταχωρητή κράτησης αποστολής (Transmit Holding register) UxTHR που αποτελεί το byte κορυφής του FIFO αποστολής. Ομοίως λαβάνονται χαρακτήρες από τον καταχωρητή λήψης (Receive Buffer register) UxRBR που είναι το byte κορυφής του FIFO λήψης. Το DLAB bit του UxLCR πρέπει να είναι ‘0’ για να μπορεί να γίνει πρόσβαση σε αυτούς τους δύο καταχωρητές. Επίσης πριν επιχειρήσουμε να διαβάσουμε ή να γράψουμε στους UxRBR ή UxTHR αντίστοιχα πρέπει να διαβάζουμε τον καταχωρητή κατάστασης γραμμής UxLSR που μας δίνει πληροφορίες αν υπάρχουν δεδομένα για να διαβαστούν από τον UxRBR ή αν ο καταχωρητής UxTHR είναι ελευθερος για να μπορέσουμε να γράψουμε τον χαρακτήρα.



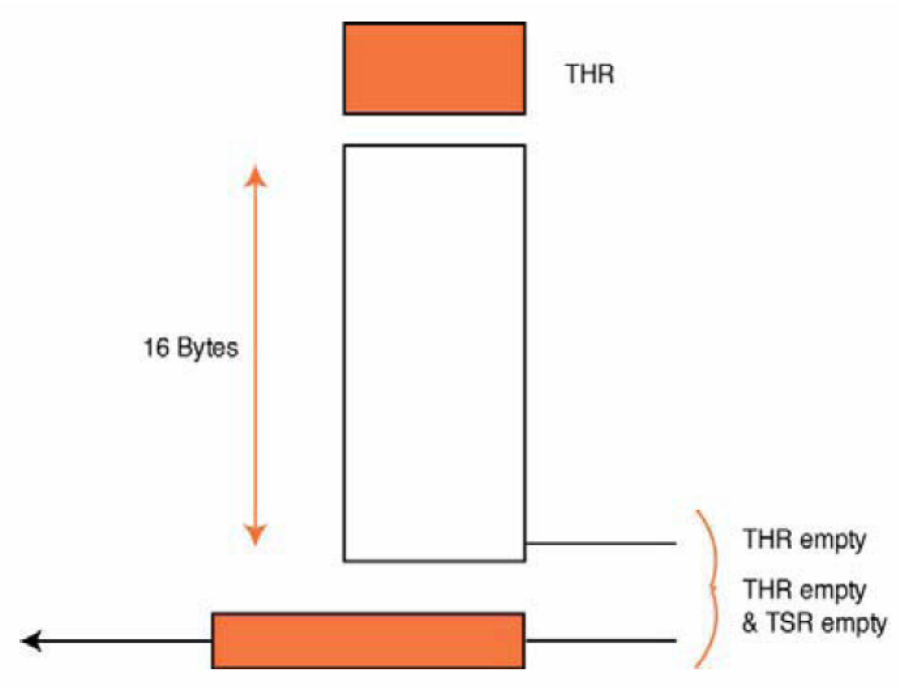
Σχήμα 4-50: Καταχωρητής κατάστασης γραμμής.

Το κάθε UART έχει μια διακοπή στον VIC αλλά κάθε διακοπή μπορεί να προκληθεί με τρεις τρόπους. Ο πρώτος τρόπος είναι αν προκληθεί κάποιο σφάλμα. Τότε μέσα στην ρουτίνα εξυπηρέτησης διακοπής μπορεί να διαβαστεί ο καταχωρητής κατάστασης UxLSR ώστε να διαπιστωθεί το σφάλμα. Ένας άλλος τρόπος δημιουργίας διακοπής είναι όταν υπάρχουν δεδομένα στον FIFO λήψης.

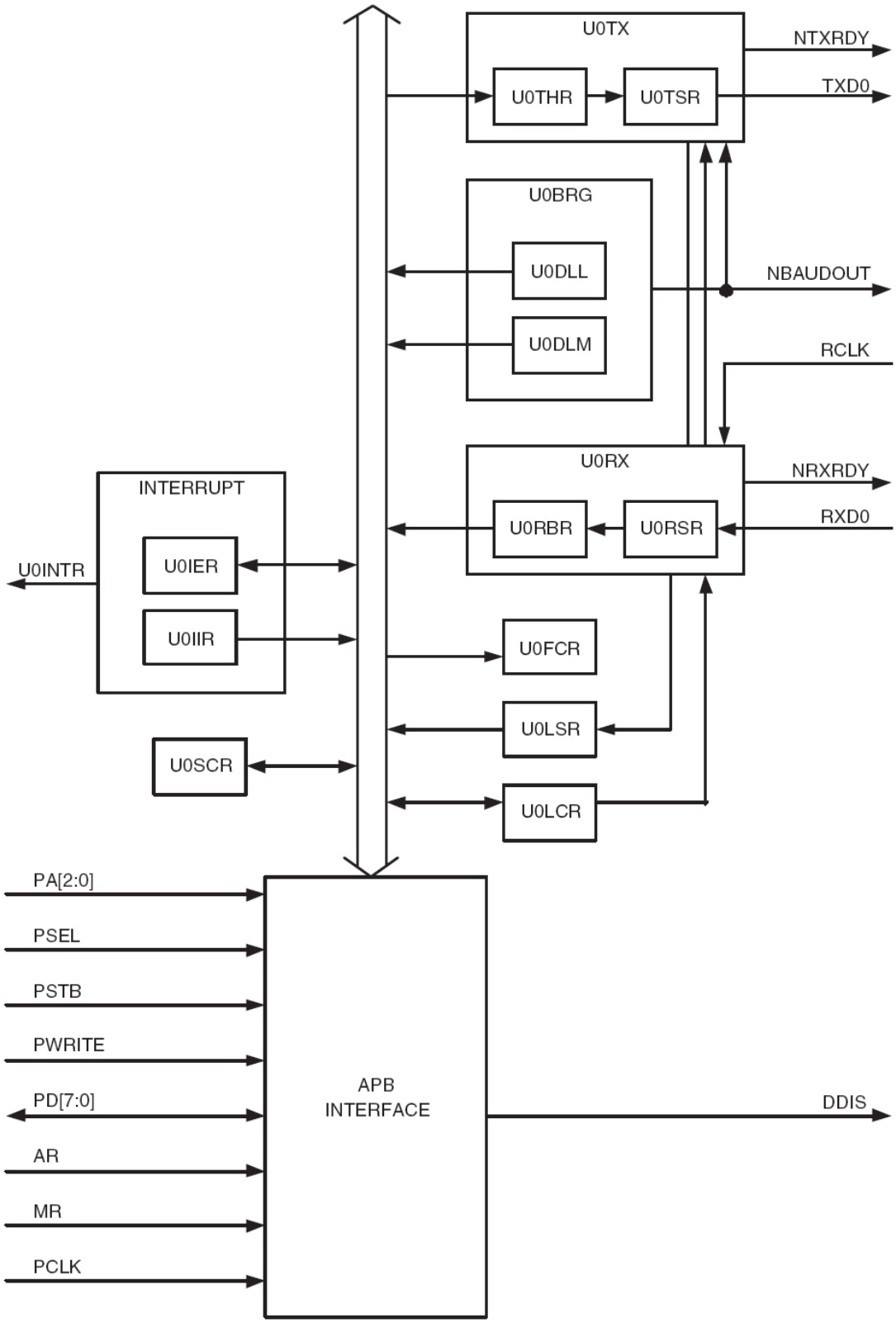


Σχήμα 4-51: Καθορισμός δημιουργίας διακοπής ανάλογα με τα byte που έλαβε ο RBR.

Ο χρήστης μπορεί να καθορίσει αν η διακοπή θα γίνει όταν θα έχουν ληφθεί 1, 4, 8 ή 14 bytes. Μέσα στην ρουτίνα εξυπηρέτησης διακοπής αυτά τα bytes μπορούν να διαβαστούν. Τέλος μια άλλη πηγή δημιουργίας διακοπής είναι όταν ο καταχωρητής αποστολής είναι άδειος. Τότε μέσα στην ρουτίνα εξυπηρέτησης διακοπής μπορούμε να γράψουμε τα δεδομένα που θέλουμε να στείλουμε.



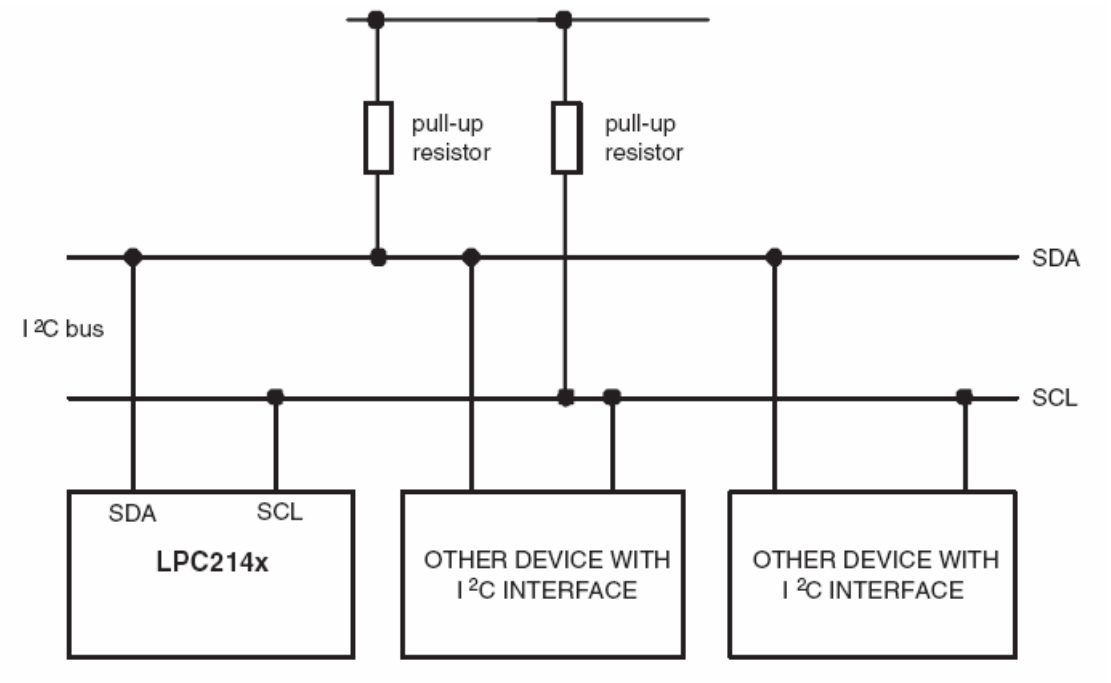
Σχήμα 4-52: Όταν ο καταχωρητής αποστολής δεδομένων είναι άδειος μπορεί να δημιουργηθεί διακοπή.



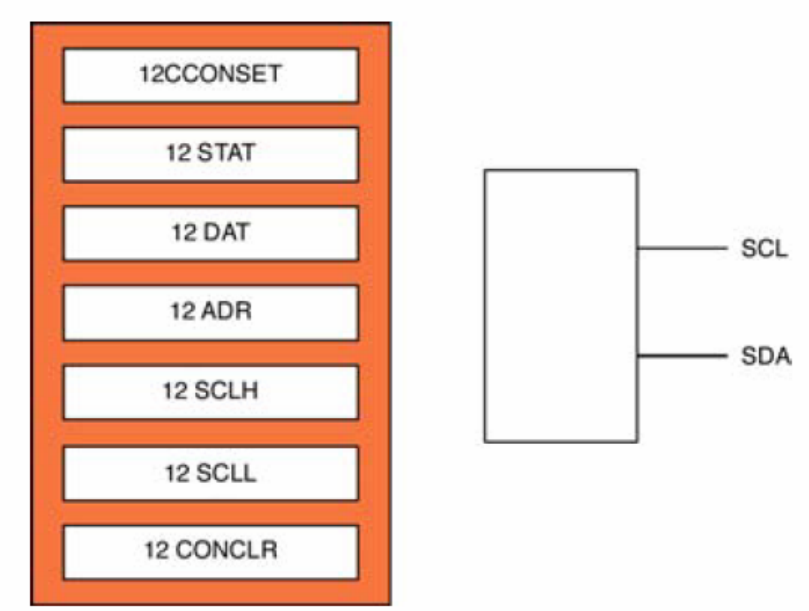
Σχλημα 4-53: Μπλοκ διάγραμμα UART0.

4.22 I^2C Διασύνδεση

Η I^2C είναι μια πολύ διαδεδομένη διασύνδεση που δημιουργήθηκε από την Philips και υποστηρίζει μεγάλο εύρος εφαρμογών όπως επικοινωνιά με μνήμες, αισθητήρες, μικροελεγκτές κ.α. Η I^2C διασύνδεση μπορεί να λειτουργήσει σε κατάσταση “αφέντης” (master) ή “δούλος” (slave) και μπορεί να φτάσει σε ταχύτητες τα 400Kbps. Τα πινς SDA και SCL αποτελούν τις I^2C γραμμές των δεδομένων και του ρολογιού αντίστοιχα και η λειτουργία I^2C πρέπει να επιλεγεί πρώτα από το pin select block.



Σχήμα 4-54: Σύνδεση συσκευών πάνω στον διάδρομο I2C.



Σχήμα 4-55: Καταχωρητές ελέγχου διασύνδεσης I2C.

Το LPC2148 έχει δύο διασυνδέσεις I^2C τα I2C0 και I2C1 οι οποίες έχουν την ίδια λειτουργία. Η I^2C διασύνδεση αποτελείται από επτά καταχωρητές. Ο καταχωρητής ελέγχου ελέγχεται από δύο καταχωρητές τους I2CxCONSET που γράφοντας '1' σε αυτόν γίνονται '1' τα αντίστοιχα bit του καταχωρητή ελέγχου και του I2CxCONCLR που γράφοντας '1' σε αυτόν γίνονται '0' τα αντίστοιχα bit του καταχωρητή ελέγχου.

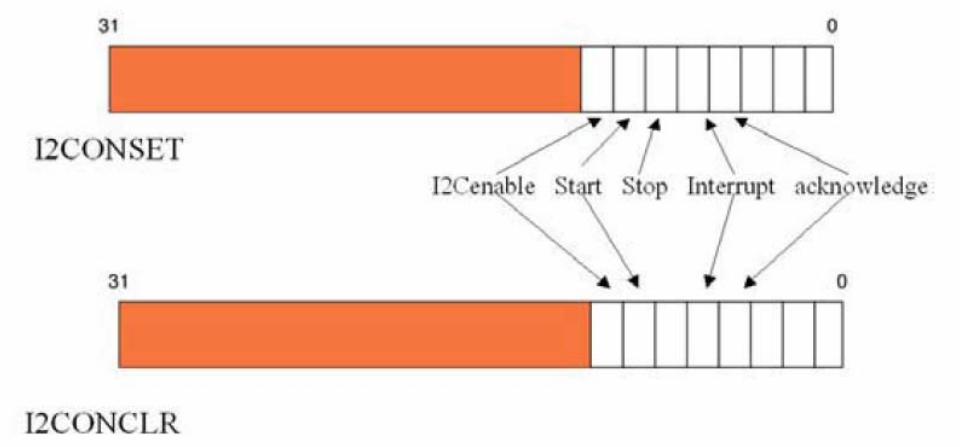
Ο ρυθμός μετάδοσης δεδομένων ρυθμίζεται από τους καταχωρητές I2CxSCLH και I2CxSCLL που διαιρούν την συχνότητα του Pclk ως εξής :

$$I^2C_{bitfrequency} = \frac{PCLK}{I2CSCLH + I2CSCLL}$$

Ο καταχωρητής κατάστασης I2CxSTAT επιστρέφει τους κωδικούς ελέγχου που σχετίζονται με διαφορετικά συμβάντα στην διασύνδεση. Ο καταχωρητής δεδομένων I2CxDAT χρησιμοποιείται για να διαβάζουμε το byte που έχει ληφθεί και για να γράφουμε το byte που θέλουμε να στείλουμε. Τέλος αν θέλουμε να χρησιμοποιήσουμε τον μικροελεγκτή σαν slave πρέπει να του ορίσουμε μια διεύθυνση η οποία γράφεται στον καταχωρητή διέυθυνσης "δούλου" (I2C slave address register) I2CxADR.

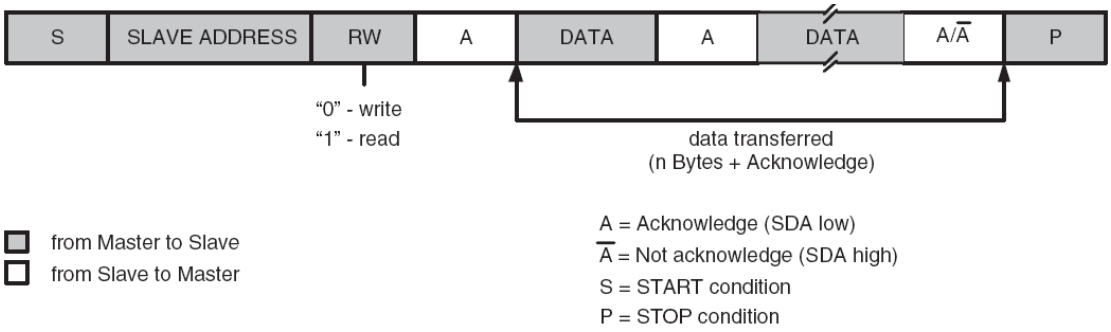
Το I^2C περιφερειακό πρέπει να προγραμματιστεί να αποκρίνεται σε κάθε συμβάν που λαμβάνει χώρα στον διάδρομο. Γι' αυτό τον λόγο το καλύτερο είναι να οδηγείται από μια διακοπή (interrupt). Η αρχικοποίηση του περιφερειακού για αυτό το σκοπό είναι η ρύθμιση του VIC με την διεύθυνση της ρουτίνας εξυπηρέτησης διακοπής και την προτεραιότητα της διακοπής I^2C. Έπειτα ρυθμίζουμε τα πινς του I^2C από GPIO σε SDA και SCL μέσω του pin select block. Στη συνέχεια καθορίζουμε τον ρυθμό μετάδοσης με τους καταχωρητές I2CSCLH και I2CSCLL.

Μόλις ρυθμιστεί το I^2C περιφερειακό, είναι έτοιμο να ανταλλάξει δεδομένα με κάποια άλλη συσκευή μέσω της I^2C διασύνδεσης. Ο έλεγχος γίνεται με τον καταχωρητή ελέγχου.

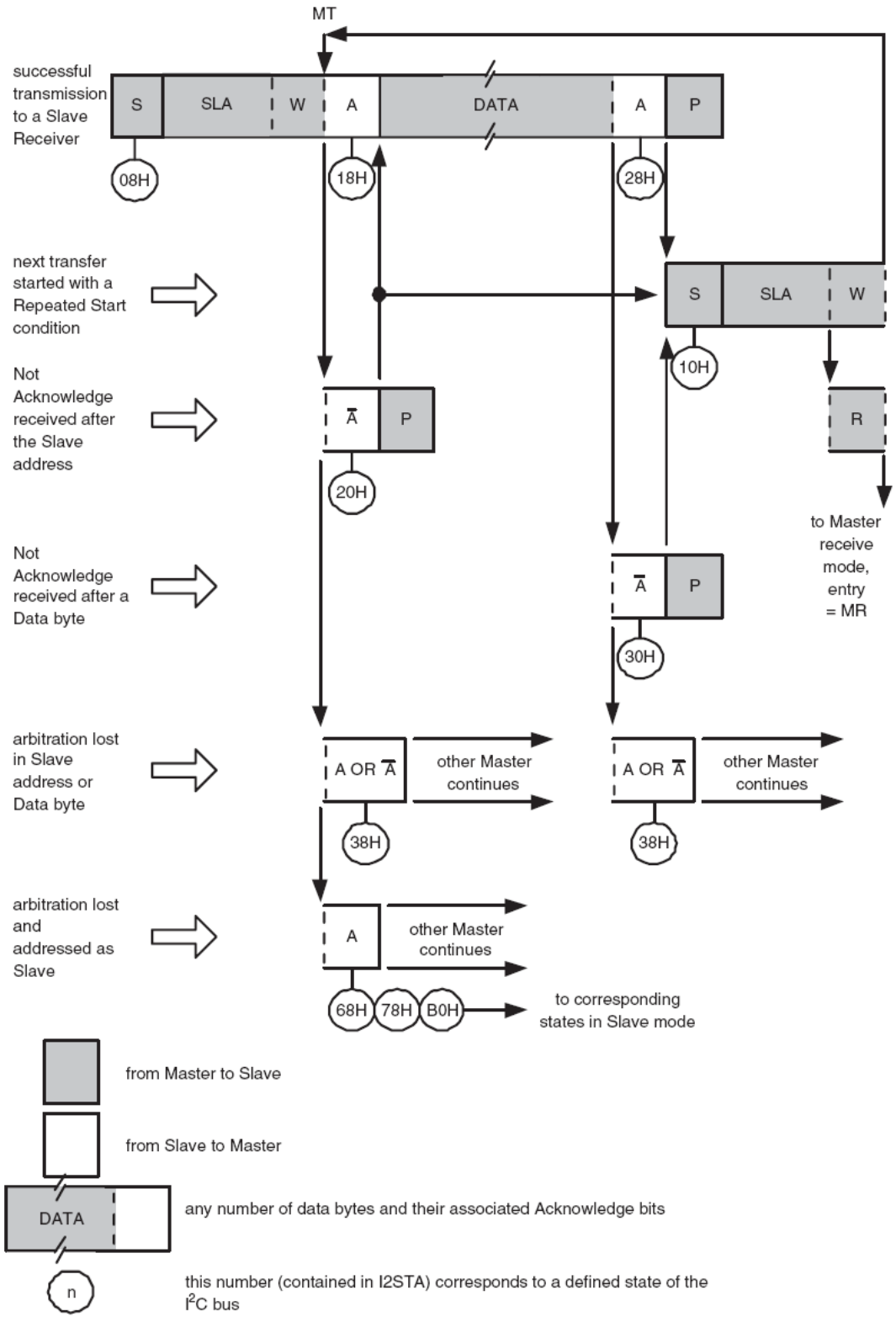


Σχήμα 4-56: Οι καταχωρητές που δίνουν τιμές στον καταχωρητή ελέγχου.

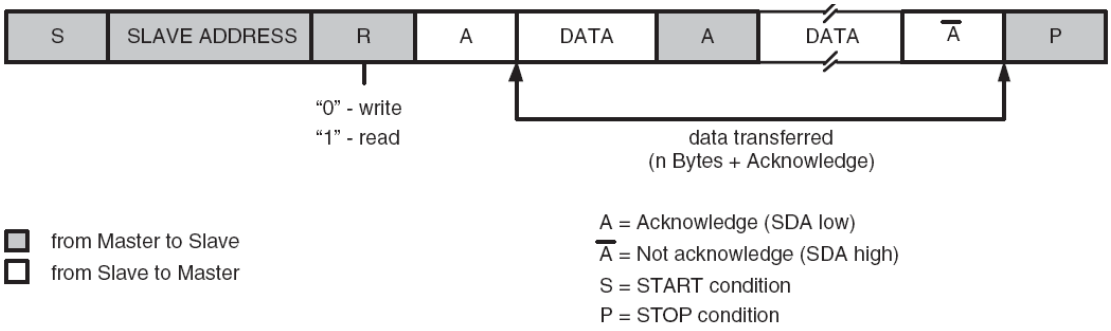
Για να λειτουργήσει το περιφερειακό σε master κατάσταση πρέπει αρχικά να ενεργοποιηθεί μεσω του bit ενεργοποίησης (enable) και το bit αναγνώρισης (acknowledge) πρέπει να γίνει '0' στον καταχωρητή ελέγχου. Αυτό αποτρέπει το I^2C περιφερειακό να αναγνωρίσει κάποιον ενδεχόμενο master και να μπει σε κατάσταση slave. Σε κατάσταση master το περιφερειακό είναι υπεύθυνο για να ξεκινήσει οποιαδήποτε επικοινωνία. Κατά τη διάρκεια μιας I^2C επικοινωνίας διάφορα γεγονότα μπορούν να συμβούν.



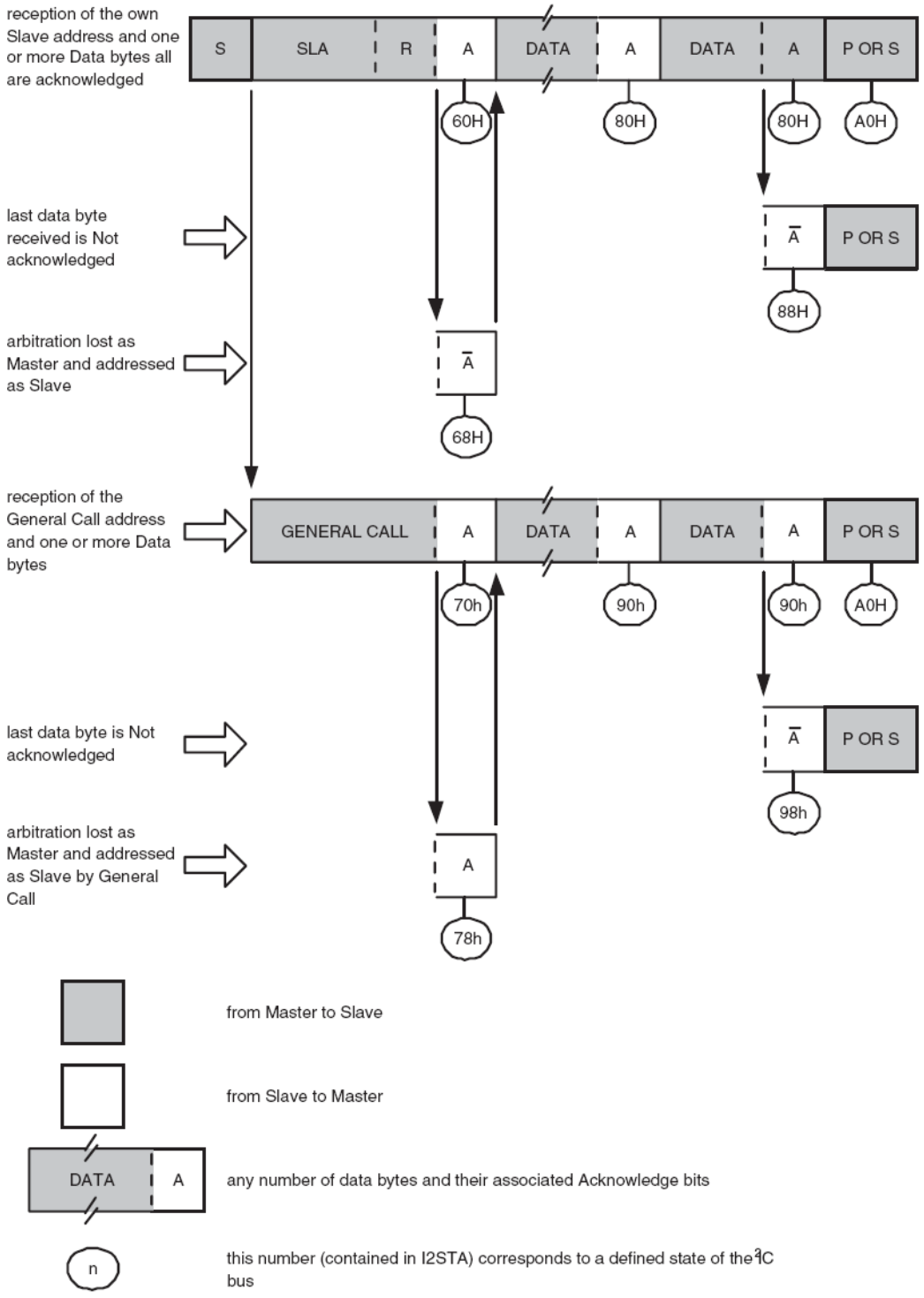
Σχήμα 4-57: Πακέτο σε κατάσταση master transmitter.



Σχήμα 4-58: Πακέτα και καταστάσεις I2C σε master trnasmitter mode.



Σχήμα 4-59: Πακέτο σε κατάσταση master receiver.



Σχήμα 4-60: Πακέτα και καταστάσεις I2C σε master receiver mode.

Αρχικά ο master πρέπει να δημιουργήσει μια κατάσταση start. Αυτό γίνεται γράφοντας ‘1’ στο start bit του καταχωρητή ελέγχου. Στην συνέχεια στέλνει την διεύθυνση του slave και το bit ανάγνωσης/εγγραφής γράφοντας ένα byte στον καταχωρητή δεδομένων I^2C. Αν ο slave λάβει σωστά το byte δημιουργεί μια κατάσταση αναγνώρισης (acknowledge) η οποία μπορεί να διαβαστεί από τον καταχωρητή κατάστασης. Στην συνέχεια ο master μπορεί να στείλει ή να

λάβει bytes επικοινωνόντας με τον slave, ωστόσο πρέπει να διαβάζονται κάθε φορά οι κωδικοί κατάστασης, από τον καταχωρητή κατάστασης I2C, που δημιουργούνται στην διασύνδεση για να εξασφαλίζεται η σωστή επικοινωνία. Οι κωδικοί αυτοί εμφανίζονται στον παρακάτω πίνακα.

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C hardware
		To/From I2DAT	To I2CON				
			STA	STO	SI	AA	
0x08	A START condition has been transmitted.	Load SLA+W	X	0	0	X	SLA+W will be transmitted; ACK bit will be received.
0x10	A repeated START condition has been transmitted.	Load SLA+W or	X	0	0	X	As above.
		Load SLA+R	X	0	0	X	SLA+W will be transmitted; the I²C block will be switched to MST/REC mode.
0x18	SLA+W has been transmitted; ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x20	SLA+W has been transmitted; NOT ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x28	Data byte in I2DAT has been transmitted; ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x30	Data byte in I2DAT has been transmitted; NOT ACK has been received.	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received.
		No I2DAT action or	1	0	0	X	Repeated START will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x38	Arbitration lost in SLA+R/W or Data bytes.	No I2DAT action or	0	0	0	X	I²C-bus will be released; not addressed slave will be entered.
		No I2DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free.

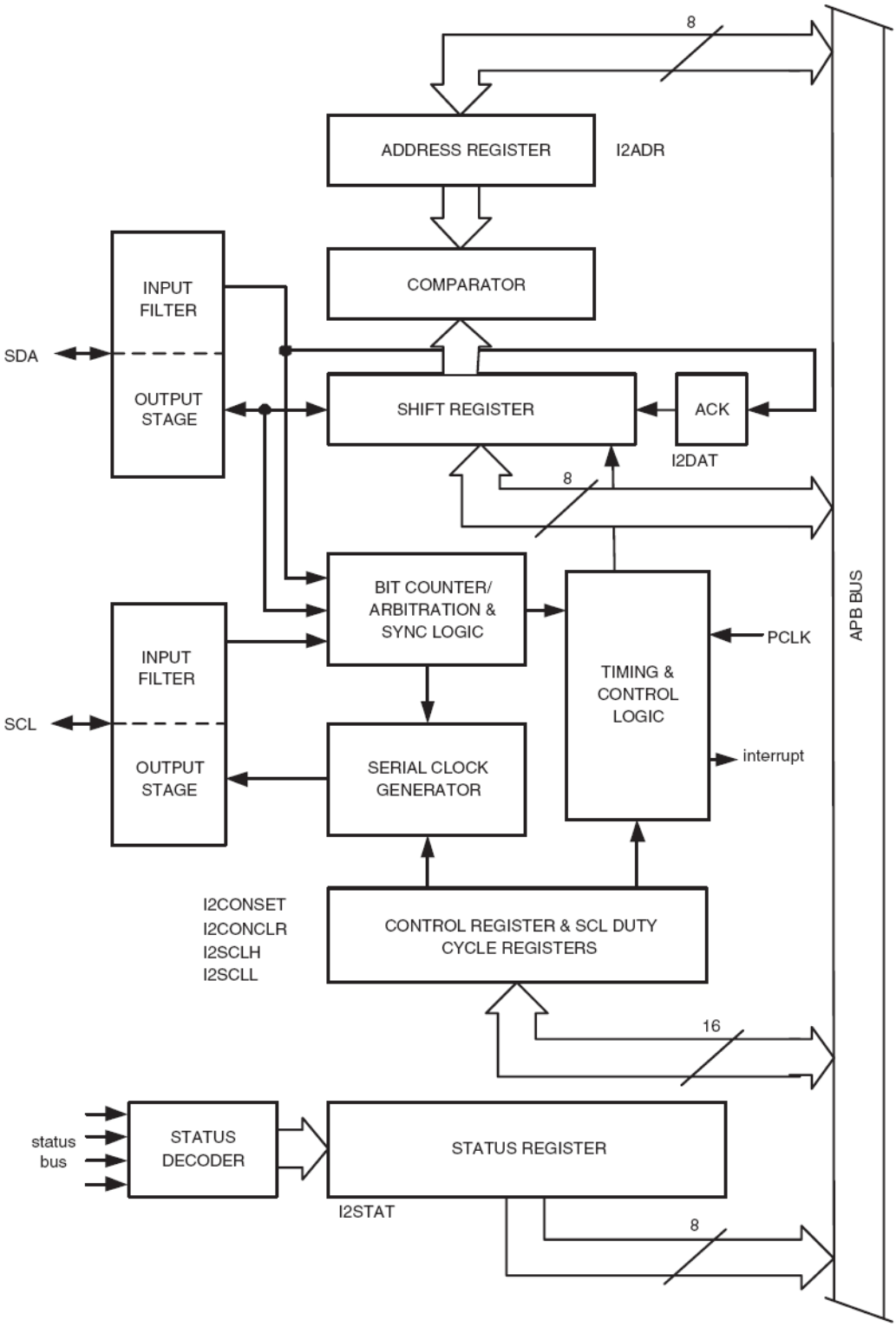
Σχήμα 4-61: Κωδικοί κατάστασης σε master transmitter mode.

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C hardware
		To/From I2DAT	To I2CON				
			STA	STO	SI	AA	
0x08	A START condition has been transmitted.	Load SLA+R	X	0	0	X	SLA+R will be transmitted; ACK bit will be received.
0x10	A repeated START condition has been transmitted.	Load SLA+R or	X	0	0	X	As above.
		Load SLA+W	X	0	0	X	SLA+W will be transmitted; the I²C block will be switched to MST/TRX mode.
0x38	Arbitration lost in NOT ACK bit.	No I2DAT action or	0	0	0	X	I²C-bus will be released; the I²C block will enter a slave mode.
		No I2DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free.
0x40	SLA+R has been transmitted; ACK has been received.	No I2DAT action or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		No I2DAT action	0	0	0	1	Data byte will be received; ACK bit will be returned.
0x48	SLA+R has been transmitted; NOT ACK has been received.	No I2DAT action or	1	0	0	X	Repeated START condition will be transmitted.
		No I2DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		No I2DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.
0x50	Data byte has been received; ACK has been returned.	Read data byte or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned.
		Read data byte	0	0	0	1	Data byte will be received; ACK bit will be returned.
0x58	Data byte has been received; NOT ACK has been returned.	Read data byte or	1	0	0	X	Repeated START condition will be transmitted.
		Read data byte or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset.
		Read data byte	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset.

Σχήμα 4-62: Κωδικοί κατάστασης σε master receiver mode.

Status Code (I2CSTAT)	Status of the I²C-bus and hardware	Application software response					Next action taken by I²C hardware
		To/From I2DAT	To I2CON				
			STA	STO	SI	AA	
0xF8	No relevant state information available; SI = 0.	No I2DAT action	No I2CON action				Wait or proceed current transfer.
0x00	Bus error during MST or selected slave modes, due to an illegal START or STOP condition. State 0x00 can also occur when interference causes the I²C block to enter an undefined state.	No I2DAT action	0	1	0	X	Only the internal hardware is affected in the MST or addressed SLV modes. In all cases, the bus is released and the I²C block is switched to the not addressed SLV mode. STO is reset.

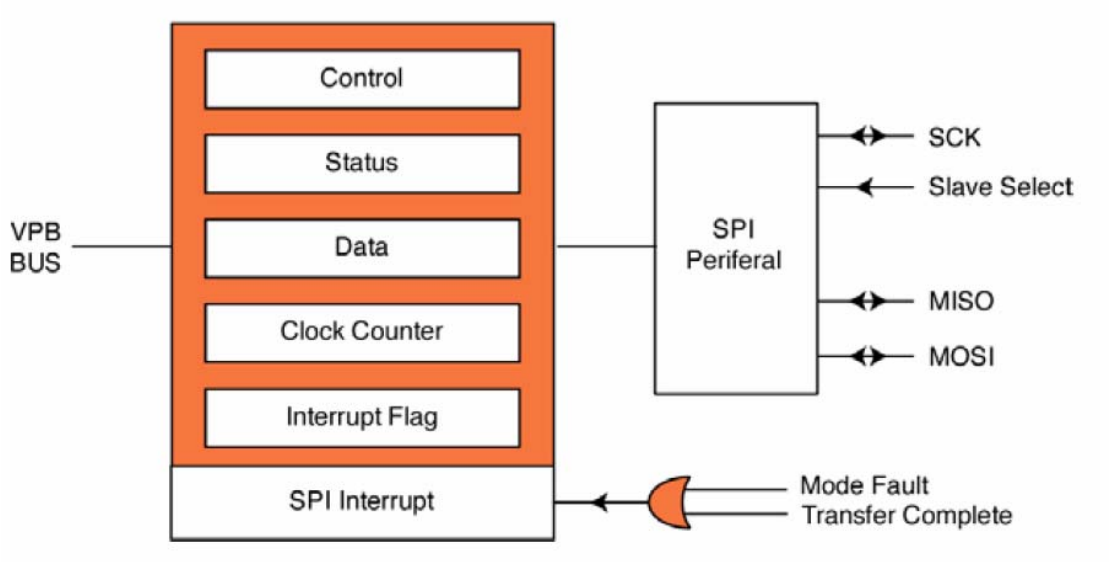
Σχήμα 4-63: Καταστάσεις μη πληροφορίας και λάθους.



Σχήμα 4-64: Μπλοκ διάγραμμα I2C περιφερειακού.

4.23 Διασύνδεση SPI

Η διασύνδεση SPI μπορεί να στείλει και να λάβει bytes μέσω του διαδρόμου SPI αλλά δεν είναι αρκετά προχωρημένη ώστε να ελέγχει τον διάδρομο. Αυτό πρέπει να γίνεται μέσω του κώδικα.

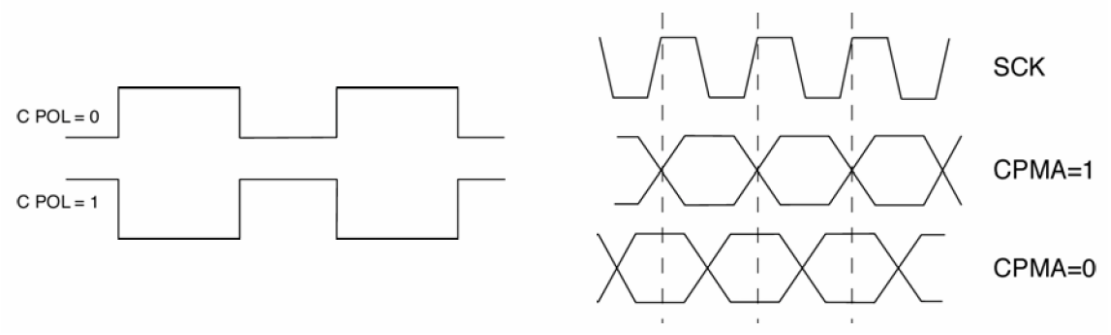


Σχήμα 4-65: Καταχωρητές διασύνδεσης SPI.

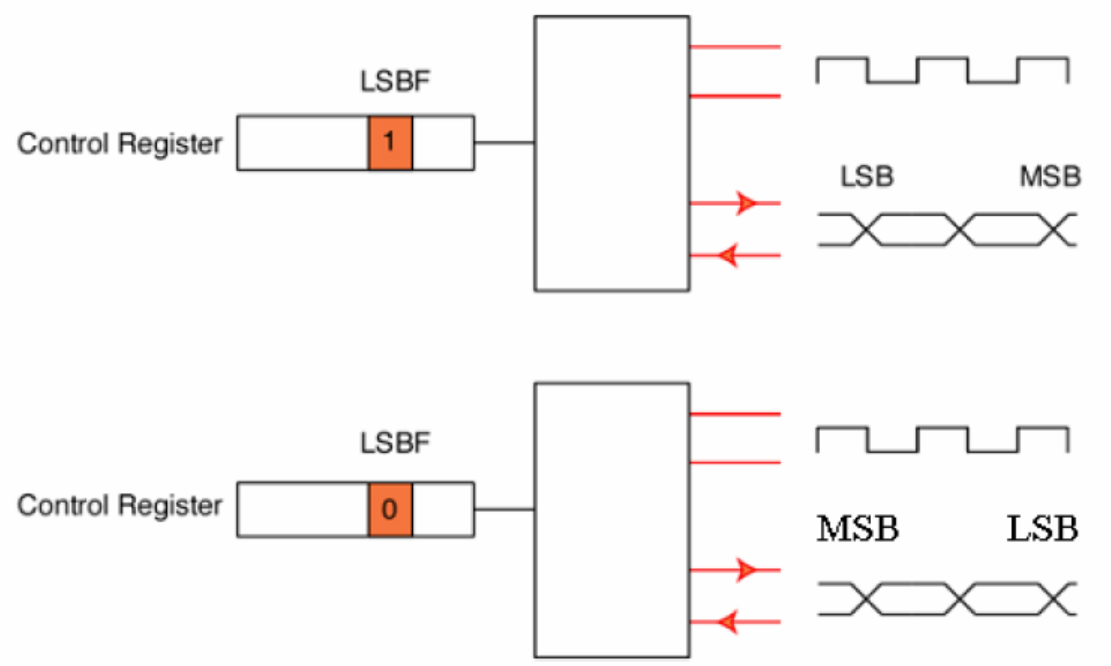
Η διασύνδεση SPI αποτελείται από τέσσερα πινς. Τα πινς σειριακού ρολογιού (SCK), δεδομένων master in / slave out (MISO), δεδομένων master out / slave in (MOSI) και επιλογής ολοκληρωμένου (CS). Η επικοινωνία γίνεται καθαρά σειριακά. Αν το SPI περιφερειακό χρησιμοποιείται σαν master πρέπει να γίνει επιλογή του ολοκληρωμένου που θέλει να επικοινωνήσει μέσω του CS το οποίο μπορεί να είναι και ένα πιν γενικής εισόδου/ εξόδου GPIO. Κάνοντας λογικό ‘0’ αυτό το πιν, ο slave καταλαβαίνει ότι αυτός έχει επιλεχθεί για επικοινωνία. Αν το περιφερειακό SPI είναι slave τότε υπάρχει συγκεκριμένο CS που πρέπει να γίνει ‘0’ για να υπάρξει επικοινωνία. Τα MISO και MOSI συνδέονται με το ολοκληρωμένο που θέλουμε να επικοινωνήσουμε και ο προσαρμογή τους εξαρτάται αν έχουμε master ή slave.

Η λειτουργία SPI μέσω του περιφερειακού επιτυγχάνεται με πέντε καταχωρητές. Ο καταχωρητής μετρητή χρονου (Clock Counter register) S0SPCCR ρυθμίζει τον ρυθμό μετάδοσης δεδομένων διαιρώντας το Pclk με την τιμή που υπάρχει σε αυτόν. Αυτός ο καταχωρητής πρέπει να έχει την ελάχιστη τιμή οκτώ. Η ρύθμιση της λειτουργίας της SPI διασύνδεσης γίνεται με τον καταχωρητή ελέγχου S0SPCR. Σε αυτόν υπάρχει το bit enable που ενεργοποιεί ή απενεργοποιεί το περιφερειακό και τα CPHA και CPOL bits που ελέγχουν την φάση του ρολογιού και την πολικότητά του σε σχέση με τα δεδομένα. Με αυτόν τον τρόπο υποστηρίζονται τέσσερις λειτουργίες SPI. Το MSTR bit καθορίζει αν το περιφερειακό θα είναι master ή slave. Το LSBF bit καθορίζει αν θα μεταφέρεται πρώτα το MSB ή το LSB του byte. Το SPIE ενεργοποιεί το SPI περιφερειακό σαν πηγή διακοπής. Τέλος το πεδίο BITS καθορίζει πόσα bits θα μεταφέρονται κάθε φορά.

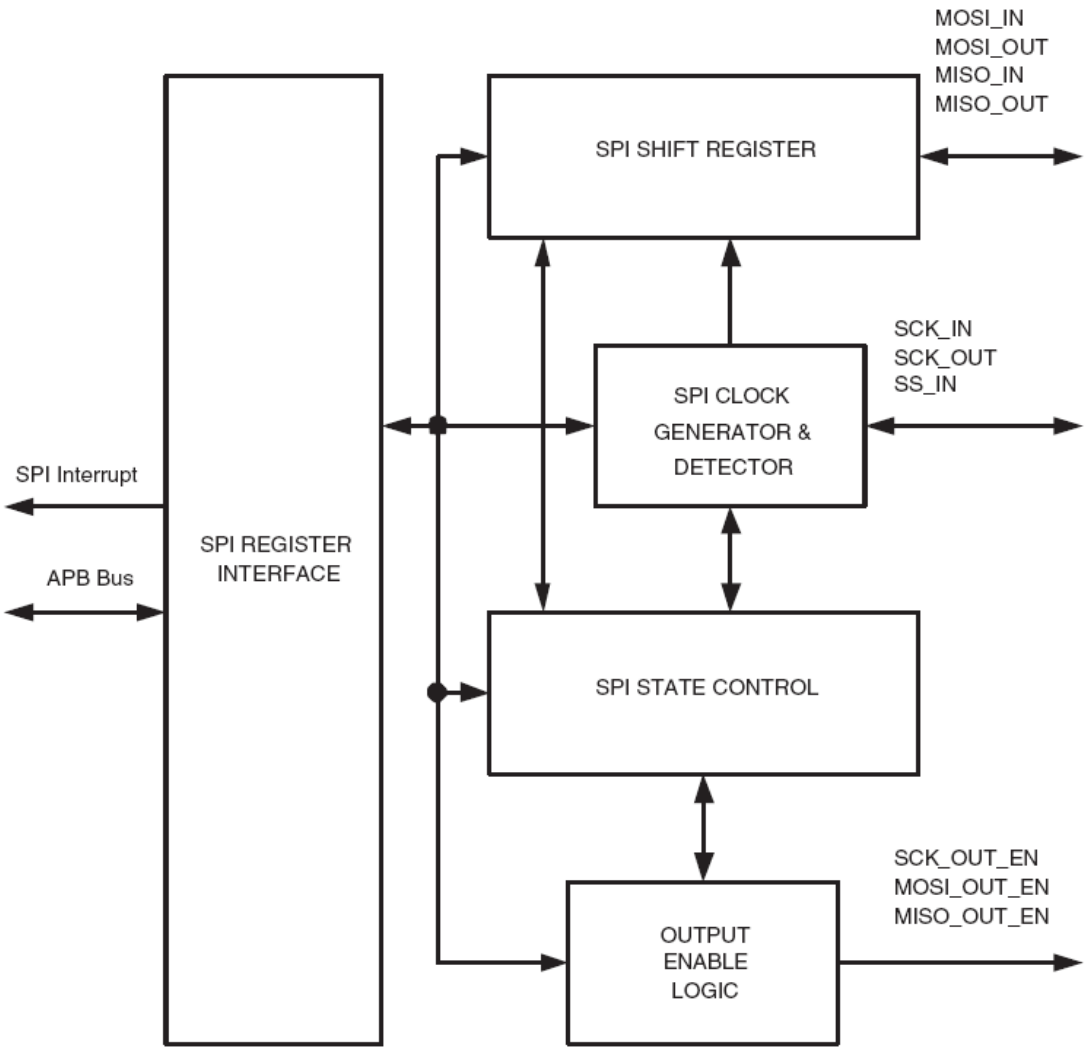
Για να γίνει επικοινωνία SPI πρέπει να καθοριστούν τα πινς του μικροελεγκτή σε λειτουργία SPI, να ρυθμιστεί ο καταχωρητής ελέγχου και στην συνέχεια να γράψουμε το byte που θέλουμε να στείλουμε στον καταχωρητή δεδομένων S0SPDR. Μετά διαβάζοντας το SPIF bit του καταχωρητή κατάστασης S0SPSR καταλαβαίνουμε ότι η μεταφορά ολοκληρώθηκε και μπορούμε να διαβάσουμε το byte που μας έστειλε η άλλη συσκευή πάλι από τον καταχωρητή δεδομένων. Η διασύνδεση SPI υποστηρίζει και λειτουργία διακοπής.



Σχήμα 4-66: Ο έλεγχος της φάσης και της πολικότητας του ρολογιού SPI.



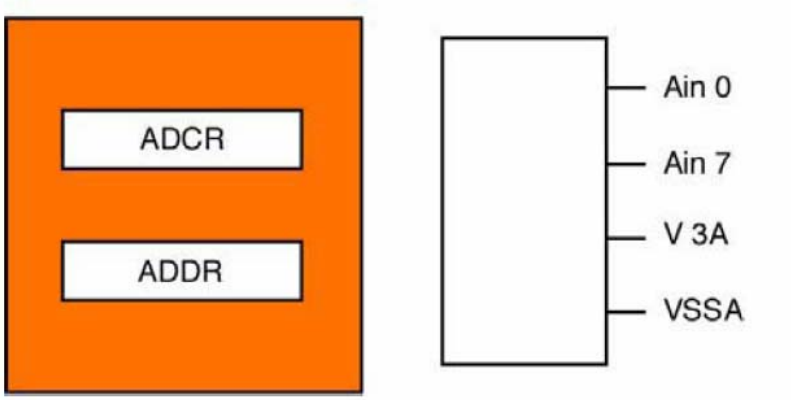
Σχήμα 4-67: Έλεγχος της φοράς του byte δεδομένων.



Σχήμα 4-68: Μπλοκ διάγραμμα του SPI περιφερειακού.

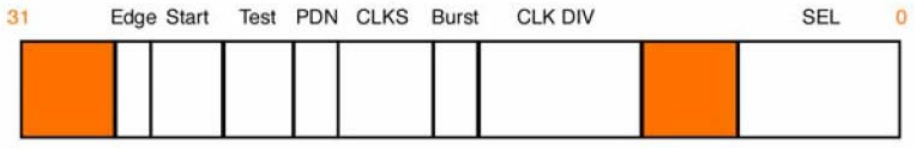
4.24 Αναλογικοψηφιακός μετατροπέας

Ο ADC που υπάρχει στους μικροελεγκτές LPC2000 έχει ακρίβεια 10bit, βασίζεται στην τεχνική διαδοχικής προσέγγισης, ο χρόνος μετατροπής είναι 2.44μsec και η μέγιστη δειγματοληψία είναι 410ksps. Γίνεται πολυπλεξία μεταξύ 6 ή 8 πινς (ADC0 και ADC1).



Σχήμα 4-69: Οι καταχωρητές και τα πινς του συστήματος ADC.

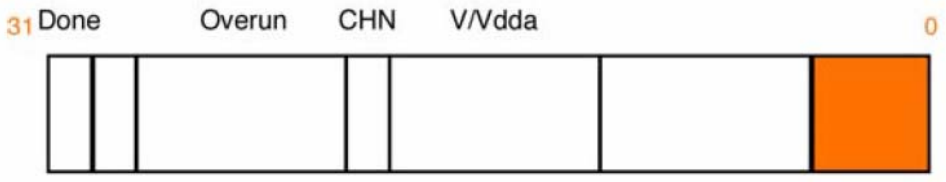
Ο καταχωρητής ελέγχου ADCR ρυθμίζει την λειτουργία του ADC και ξεκινά την διαδικασία δειγματοληψίας. Αρχικά, ρυθμίζουμε το ρολόι του περιφερειακού διαιρώντας το Pclk ώστε να έχουμε συχνότητα 4.5MHz ή την πιο κοντινή τιμή σε αυτή.



Σχήμα 4-70: Ο καταχωρητής ελέγχου ADCR.

Το Pclk διαιρείται με την τιμή του πεδίου CLKDIV συν ένα. Το PDN bit καθορίζει την κατάσταση ενέργειας του ADC και πριν την λειτουργία του ADC πρέπει να ενεργοποιηθεί. Τα πινς του ADC μπορούν να είναι GPIO κατά τη δειγματοληψία αλλά για μεγαλύτερη ακρίβεια είναι καλύτερο να έχουν επιλεχθεί μόνο για την λειτουργία ADC.

Η ακρίβεια της μετατροπής μπορεί να καθοριστεί από το CLKS πεδίο που ρυθμίζει την ακρίβεια από 10bits έως 3bits. Η ακρίβεια είναι ίση με τον αριθμό των κύκλων μετατροπής πλην ένα. Έτσι για 10bit ακρίβεια χρειάζονται 11 κύκλοι ενώ για 3bit χρειάζονται 4 κύκλοι ρολογιού. Υπάρχουν δύο καταστάσεις μετατροπής ADC. Η πρώτη είναι η κατάσταση υλικού (hardware) που επιτρέπει την επιλογή κάποιου αριθμού καναλιών και ύστερα την εκκίνηση του ADC. Οι μετατροπές γίνονται διαδοχικά σε κάθε κανάλι μέχρι να σταματήσουμε τον μετατροπέα. Στο τέλος κάθε μετατροπής τα αποτέλεσματα είναι διαθέσιμα στους καταχωρητές δεδομένων ADDRx.



Σχήμα 4-71: Ο καταχωρητής δεδομένων ADDR.

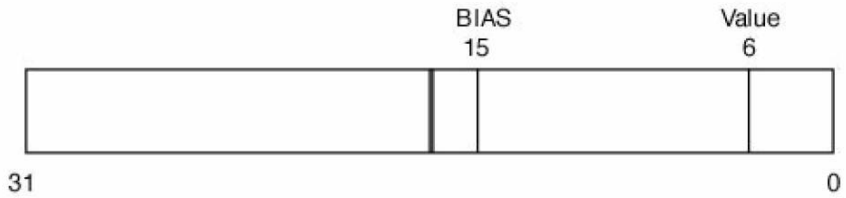
Στους καταχωρητές δεδομένων υπάρχει το DONE bit που δείχνει πότε μια μετατροπή έχει ολοκληρωθεί και μπορεί να δημιουργήσει και διακοπή. Το

αποτέλεσμα αποθηκεύεται στο RESULT πεδίο σαν μια αναλογία της τάσης στον κανάλι προς την τάση στο πιν αναλογικής τροφοδοσίας (V/Vdda). Το OVERUN bit δείχνει αν το προηγούμενο αποτέλεσμα δεν πρόλαβε να διαβαστεί και αν η καινούρια μετατροπή το αντικατέστησε.

Η δεύτερη κατάσταση είναι του λογισμικού (software). Σε αυτή, το κανάλι επιλέγεται μέσω των SEL bits και η εκκίνηση της μετατροπής γίνεται γράφοντας 0x01 στον πεδίο START. Με αυτό τον τρόπο ο ADC κάνει μια μετατροπή στο κανάλι και αποθηκεύει το αποτέλεσμα στον ADDR καταχωρητή. Η ολοκλήρωση της μετατροπής μπορεί να δημιουργήσει διακοπή ή μπορούμε να το καταλάβουμε διαβάζοντας το DONE bit του ADDR. Επίσης μπορεί να γίνει έναρξη μετατροπής με την δημιουργία συμβάντος ισοτιμίας στου μετρητές του μικροελεγκτή.

4.25 Μετατροπέας ψηφιακού σήματος σε αναλογικό

Ο LPC2148 μικροελεγκτής έχει έναν ψηφιακοαναλογικό μετατροπέα (Digital to Analog Converter) DAC ο οποίος έχει ακρίβεια 10bits. Η λειτουργία του γίνεται μεσω του καταχωρητή DACR. Ο DACR έχει ένα πεδίο VALUE των 10bits και ένα BIAS bit.



Σχήμα 4-72: Ο καταχωρητής DACR.

Αρχικά πρέπει να επιλέξουμε το πιν P0.24 ώς λειτουργία AOUT που είναι και η έξοδος του DAC. Μόλις ενεργοποιηθεί και το περιφερειακό η μετατροπή μπορεί να αρχίσει γράφοντας την τιμή που θέλουμε στο πεδίο VALUE του DACR. Ο χρόνος μετατροπής εξαρτάται από την τιμή του bit BIAS. Αν έχουμε γράψει ‘1’ τότε ο χρόνος μετατροπής είναι 2.5μs και η έξοδος μπορεί να οδηγήσει μέχρι 700μA. Αν έχουμε γράψει ‘0’ τότε ο χρόνος μετατροπής είναι 1μs αλλά η έξοδος οδηγεί μέχρι 350μA. Ωστόσο ο συνολικός χρόνος ηρεμίας της μετατροπής εξαρτάται από την εξωτερική αντίσταση. Η τάση εξόδου της AOUT δίνεται από τον τύπο AOUT= (VALUE/1024)xVREF. Το VREF είναι το πιν της τάσης αναφοράς για την μετατροπή. Επίσης η τροφοδοσία του μετατροπέα γίνεται από τα VDDA και VSSA πινς.

4.26 Υπόλοιπα περιφερειακά

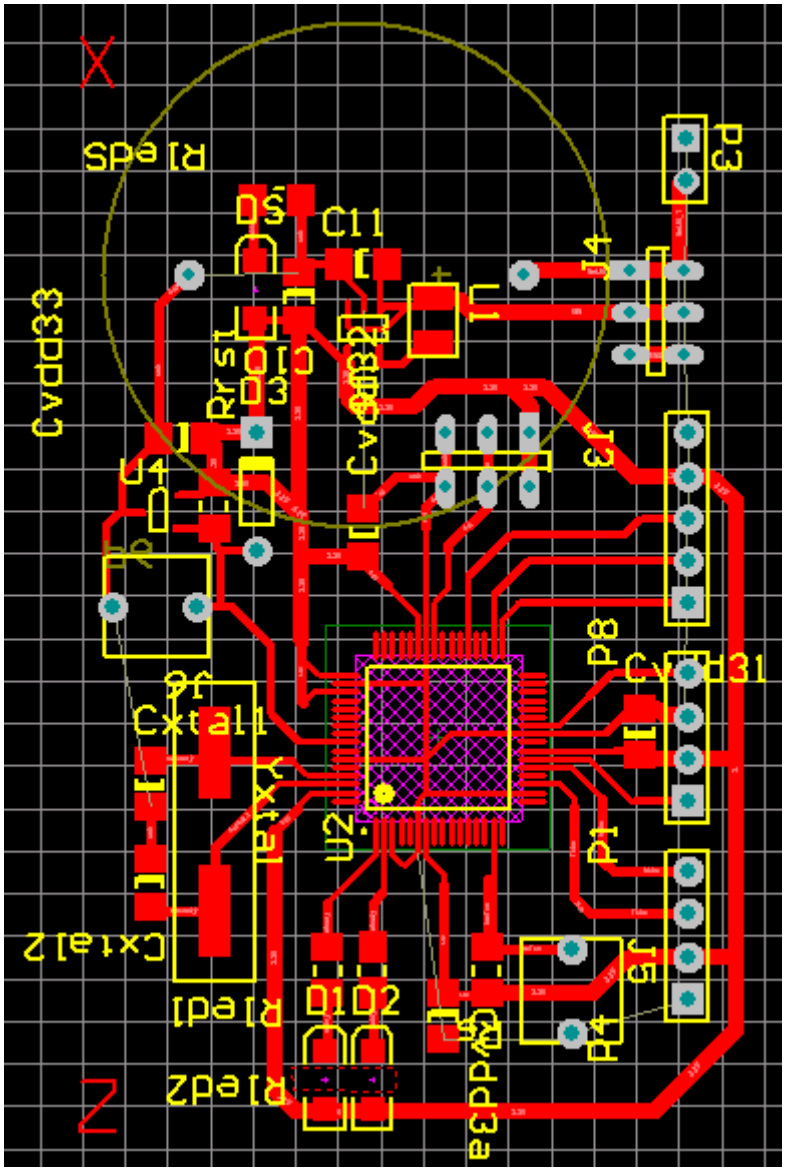
Ο μικροελεγκτής LPC2148 έχει και άλλα περιφερειακά όπως η διασύνδεση USB η οποία κυρίως χρησιμοποιείται για επικοινωνία με ηλεκτρονικό υπολογιστή, ο διαμορφωτής διάρκειας παλμών PWM (Pulse Width Modulator) ο οποίος μπορεί να χρησιμοποιηθεί για τον έλεγχο κάποιου κινητήρα ή της διαύγειας ενός led και το περιφερειακό χρονομετρητής επαγρύπνησης (watchdog timer) το οποίο ελέγχει ανά τακτά χρονικά διαστήματα έναν μετρητή εξασφαλίζοντας τη σωστή λειτουργία του μικροελεγκτή και ότι ο κώδικας δεν έχει ξεφύγει από την προγραμματισμένη λειτουργία του.

ΚΕΦΑΛΑΙΟ 5

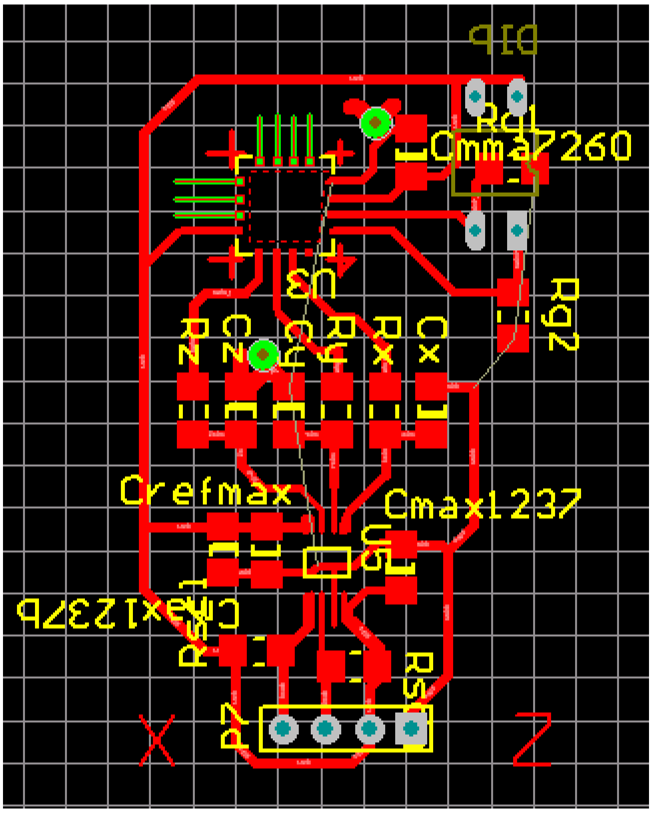
Κατασκευή συστήματος

5.1 Σχεδίαση τυπωμένων πλακετών

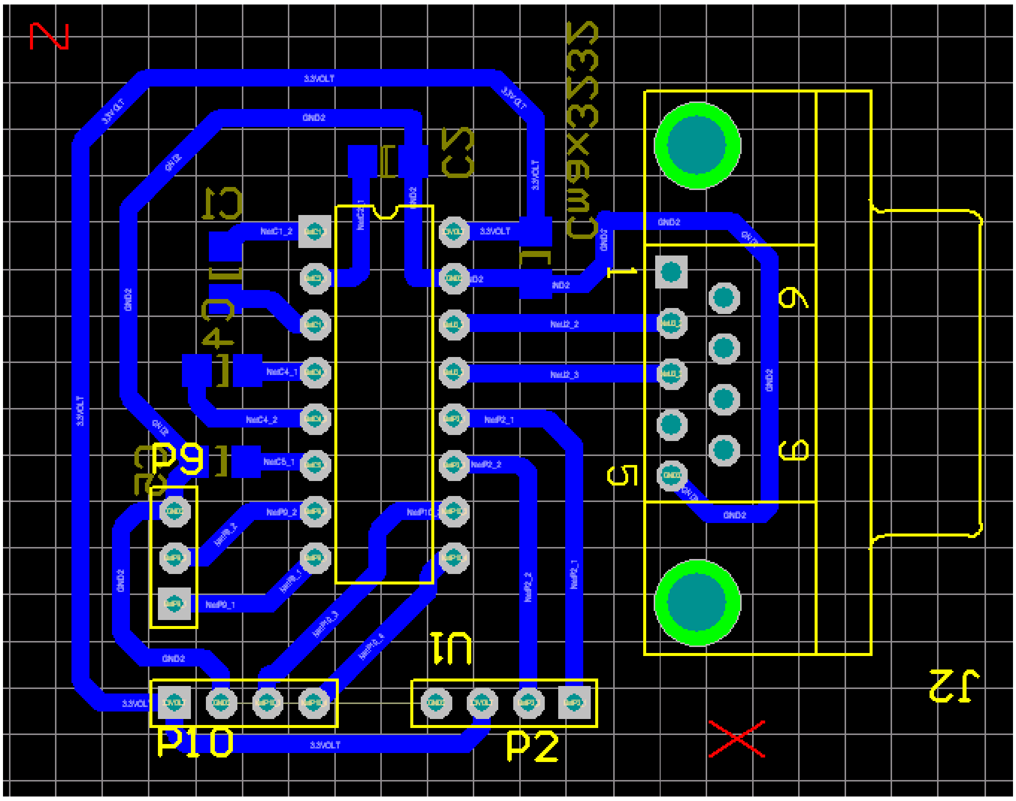
Το σύστημα σχεδιάστηκε αρχικά σε ειδικό πρόγραμμα σχεδίασης πλακετών. Πρώτα έγιναν τα σχηματικά διαγράμματα που παρουσιάστηκαν στο κεφάλαιο παρουσίασης του συστήματος και στην συνέχεια δημιουργήθηκαν τα διαγράμματα πάνω και κάτω όψης των πλακετών.



Σχήμα 5-1: Η πάνω όψη του συστήματος μικροελεγκτή στο πρόγραμμα σχεδίασης τυπωμένων πλακετών.



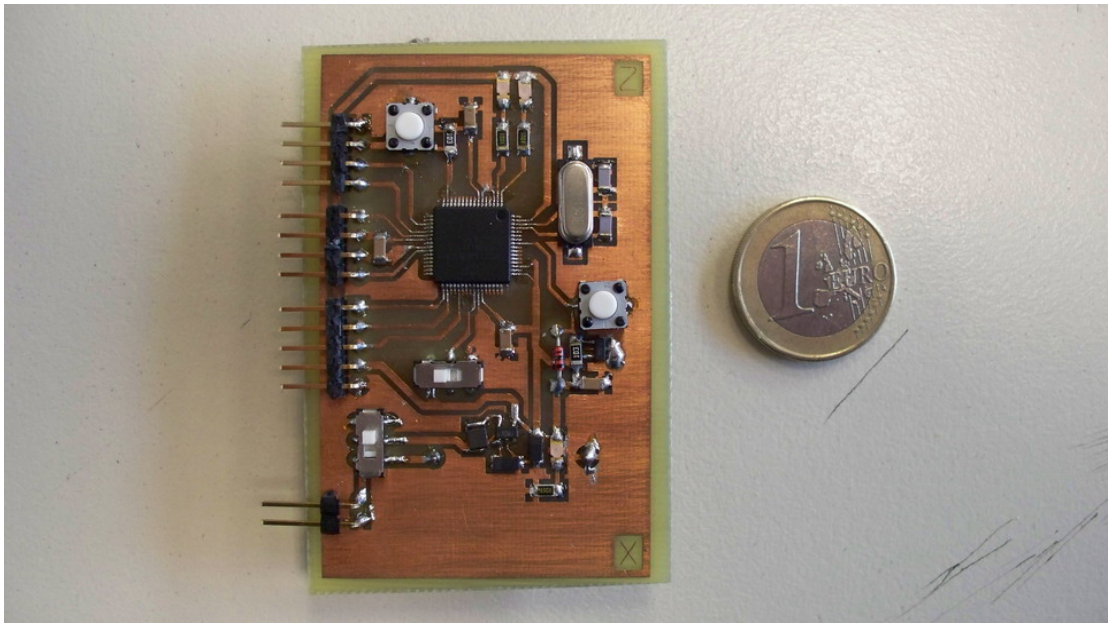
Σχήμα 5-2: Η πάνω όψη του συστήματος επιταχυνσιομέτρου στο πρόγραμμα σχεδίασης τυπωμένων πλακετών.



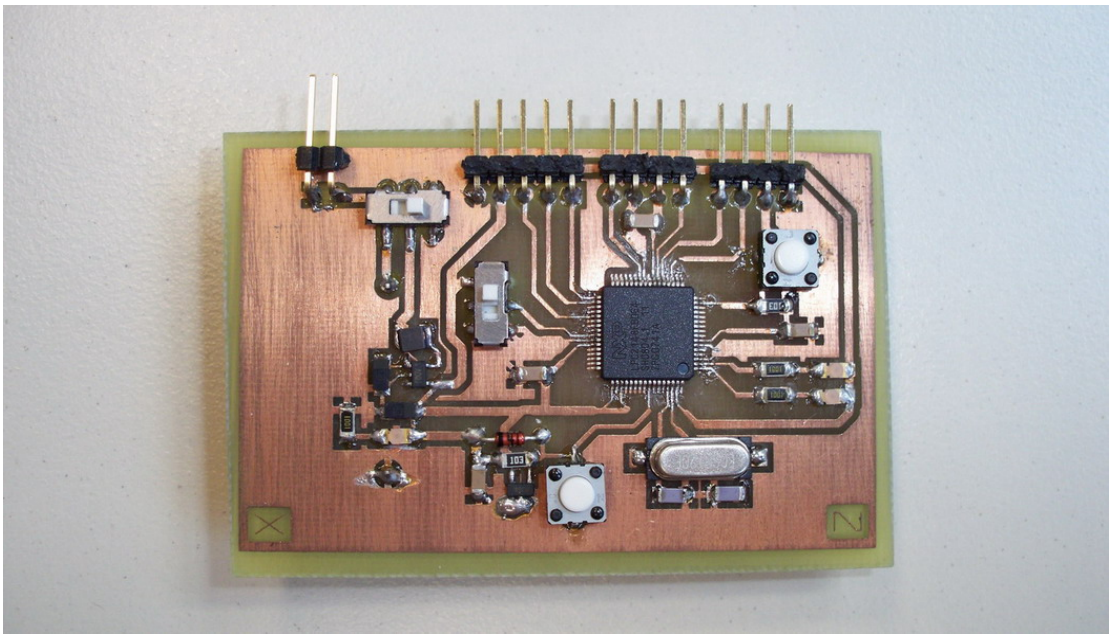
Σχήμα 5-3: Η κάτω όψη του συστήματος μετατροπής σειριακών σημάτων στο πρόγραμμα σχεδίασης τυπωμένων πλακετών.

5.2 Κατασκευή πλακετών

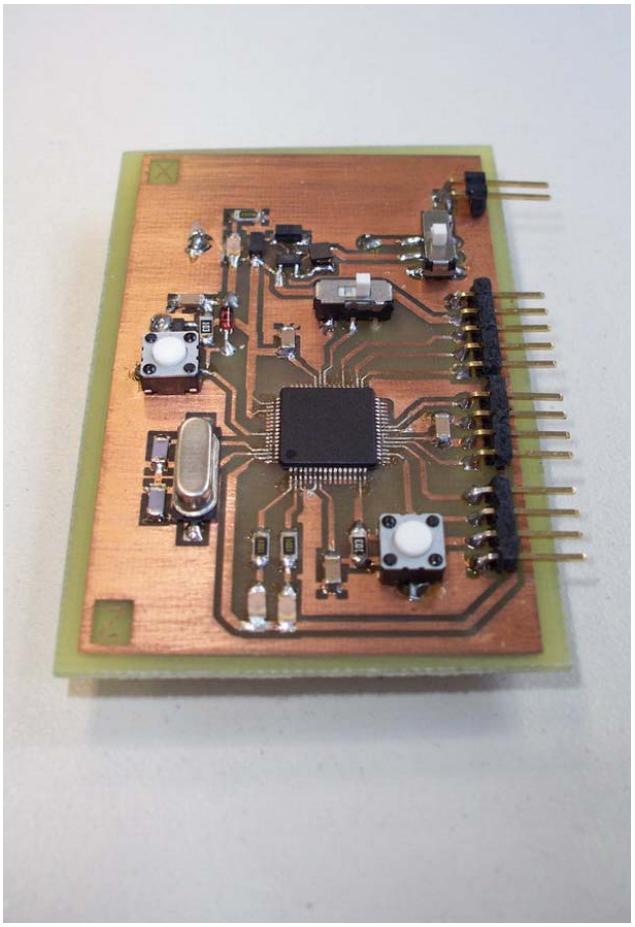
Οι πλακέτες κατασκευάστηκαν στο εργαστήριο αισθητήρων του Εθνικού Μετσόβιου Πολυτεχνείου. Πρώτα τυπώθηκαν σε διαφάνειες τα κυκλώματα και στην συνέχεια τοποθετήθηκαν πάνω στις φωτοευαίσθητες πλακέτες δύο όψεων. Οι πλακέτες εκτέθηκαν σε υπεριώδες φως και μετά χρησιμοποιήθηκαν χημικά για να γίνει η αποχάλκωση.



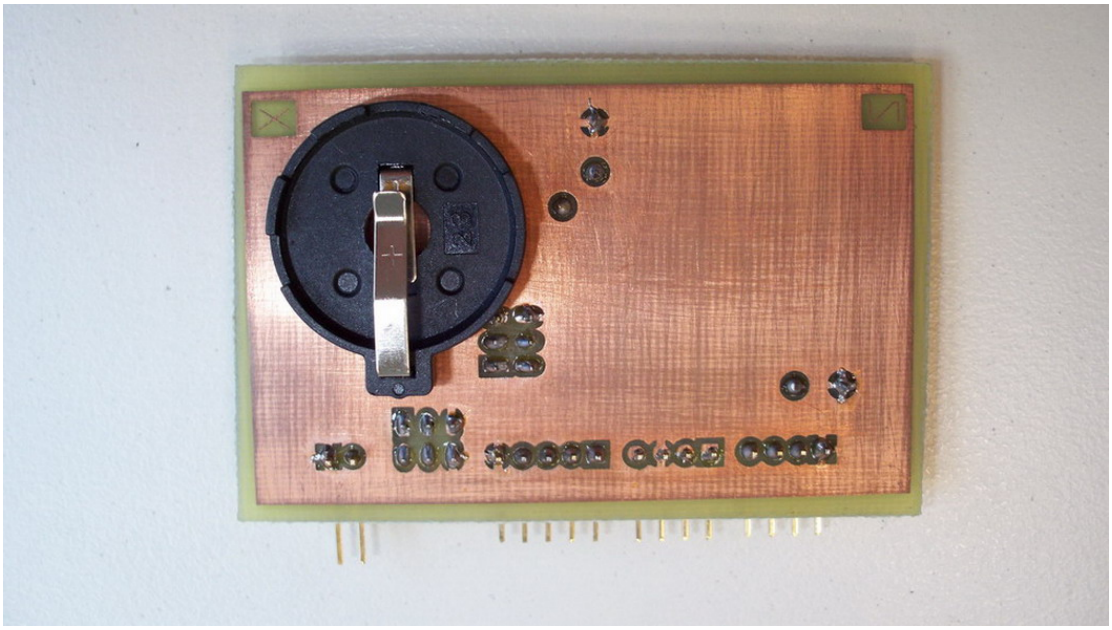
Σχήμα 5-4: Η πάνω όψη της πλακέτας του συστήματος μικροελεγκτή συγκριτικά με ένα νόμισμα του ενός ευρώ.



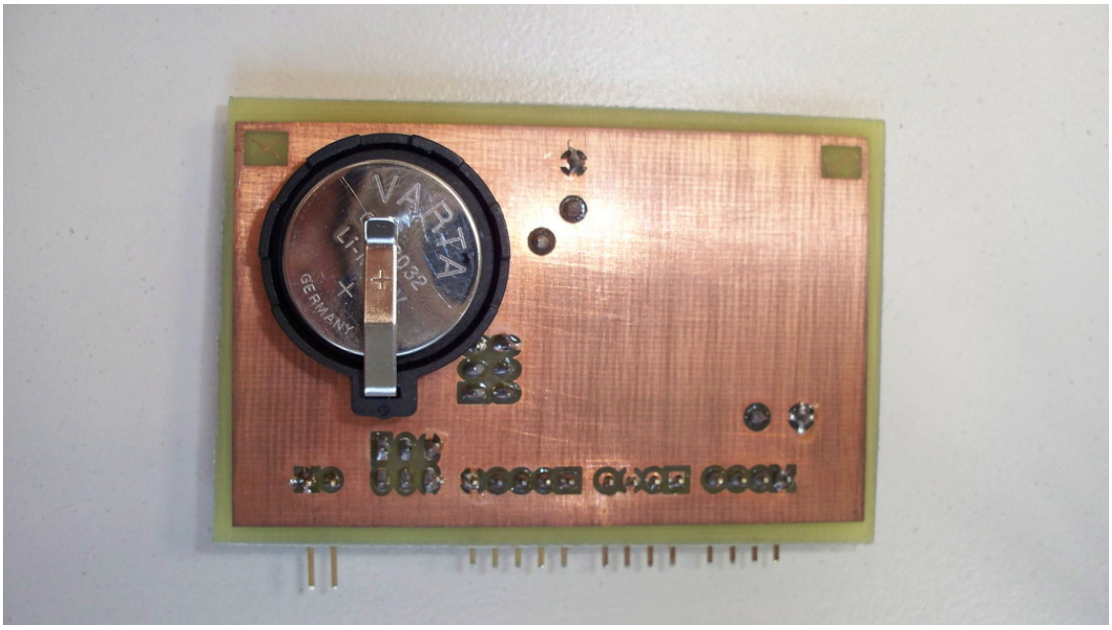
Σχήμα 5-5: Η πάνω όψη της πλακέτας του υποσυστήματος μικροελεγκτή.



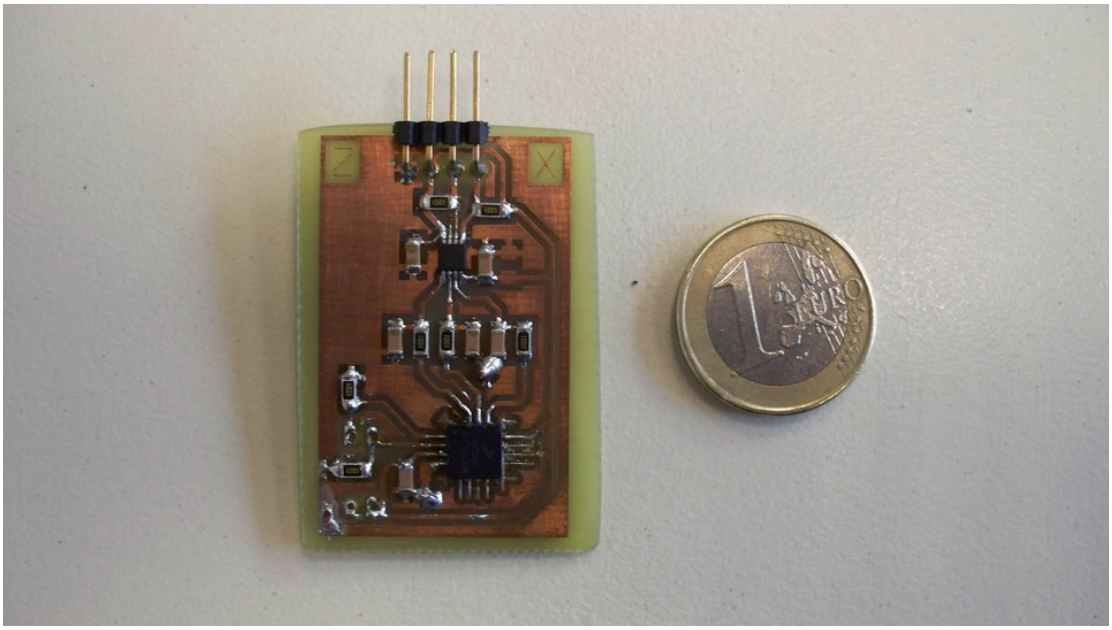
Σχήμα 5-6: Η πάνω όψη της πλακέτας του υποσυστήματος μικροελεγκτή.



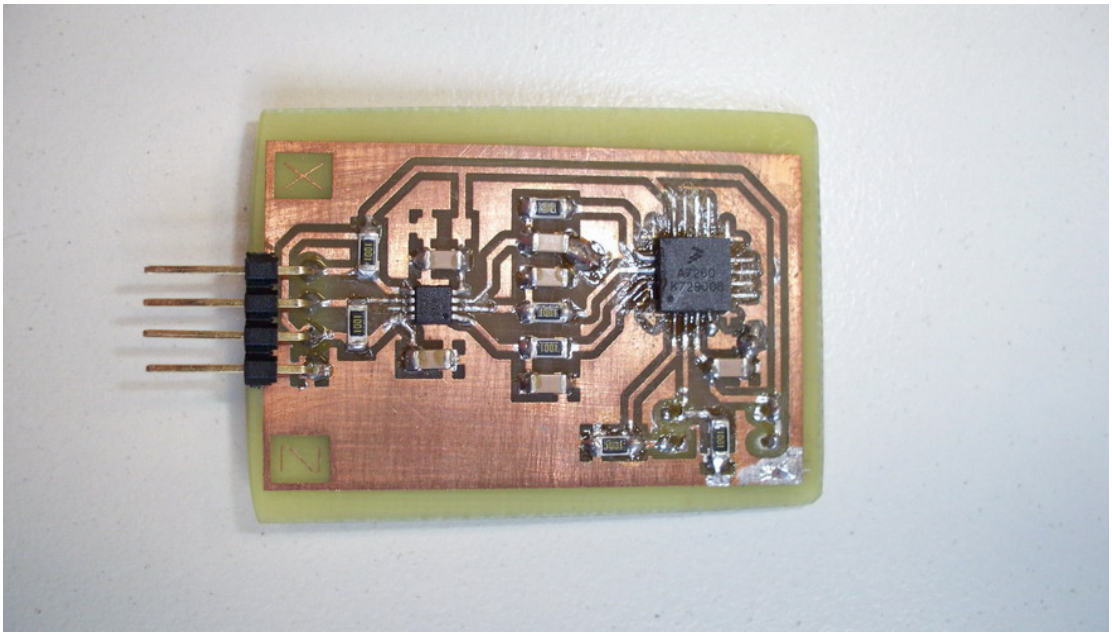
Σχήμα 5-7: Η κάτω όψη της πλακέτας του υποσυστήματος μικροελεγκτή.



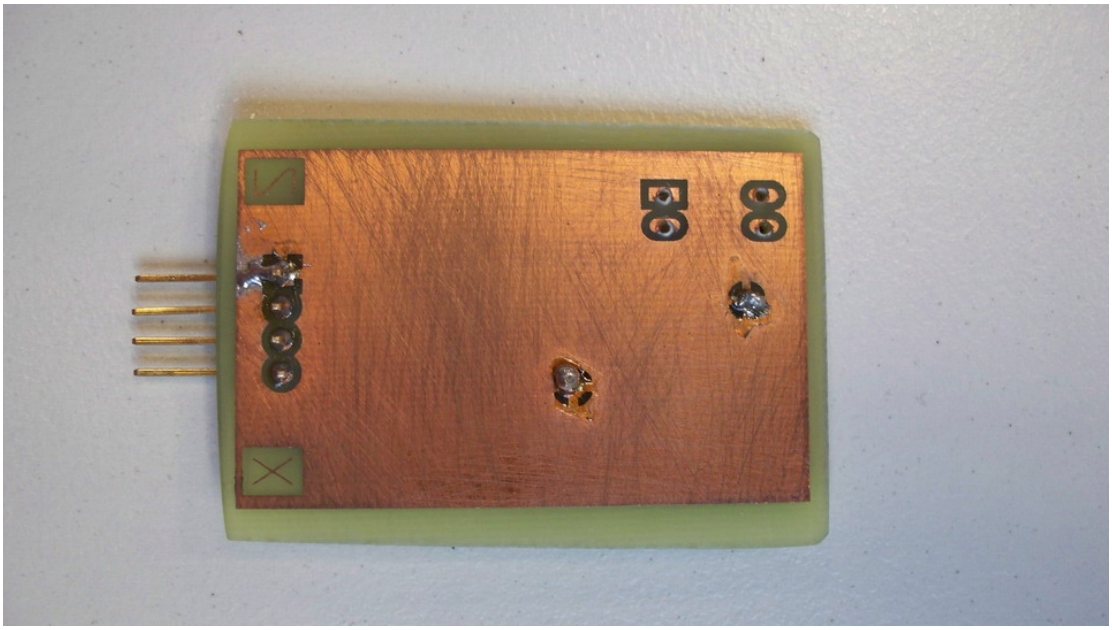
Σχήμα 5-8: Η κάτω όψη της πλακέτας του υποσυστήματος μικροελεγκτή με τοποθετημένη της μπαταρία λιθίου στην υποδοχή της.



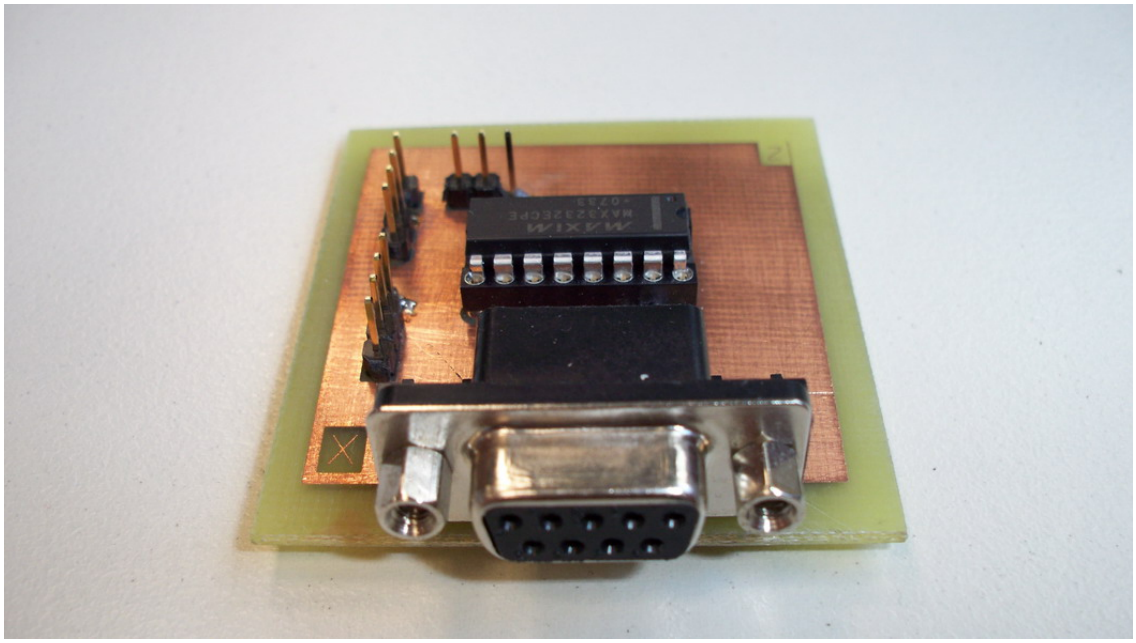
Σχήμα 5-9: Η πάνω όψη της πλακέτας του συστήματος επιταχυνσιομέτρου συγκριτικά με ένα νόμισμα του ενός ευρώ.



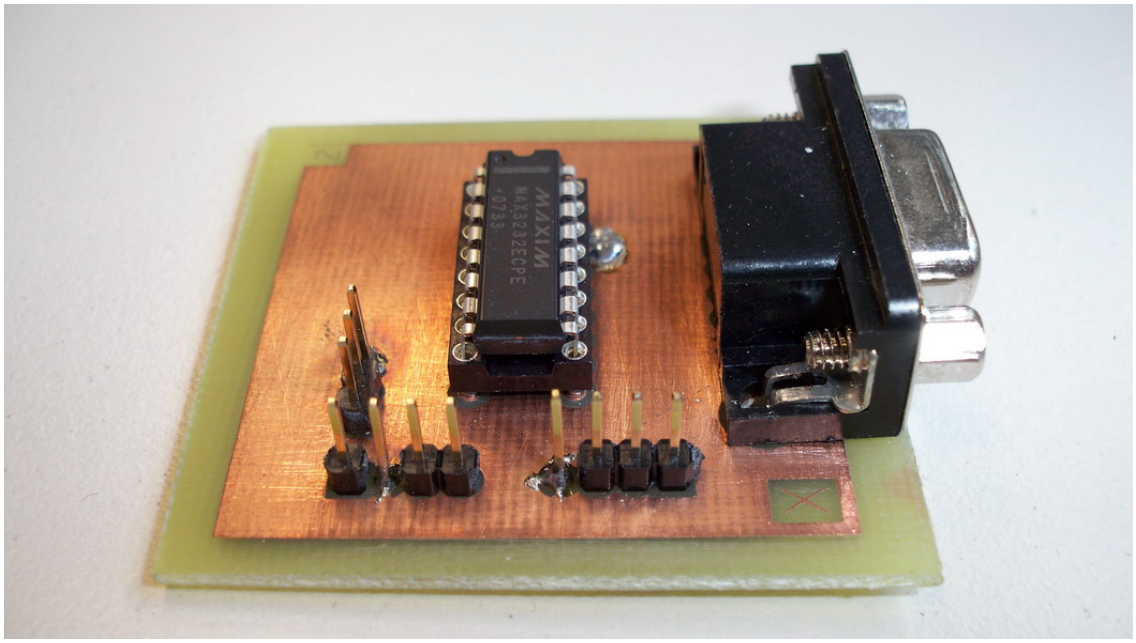
Σχήμα 5-10: Η πάνω όψη της πλακέτας του συστήματος επιταχυνσιομέτρου.



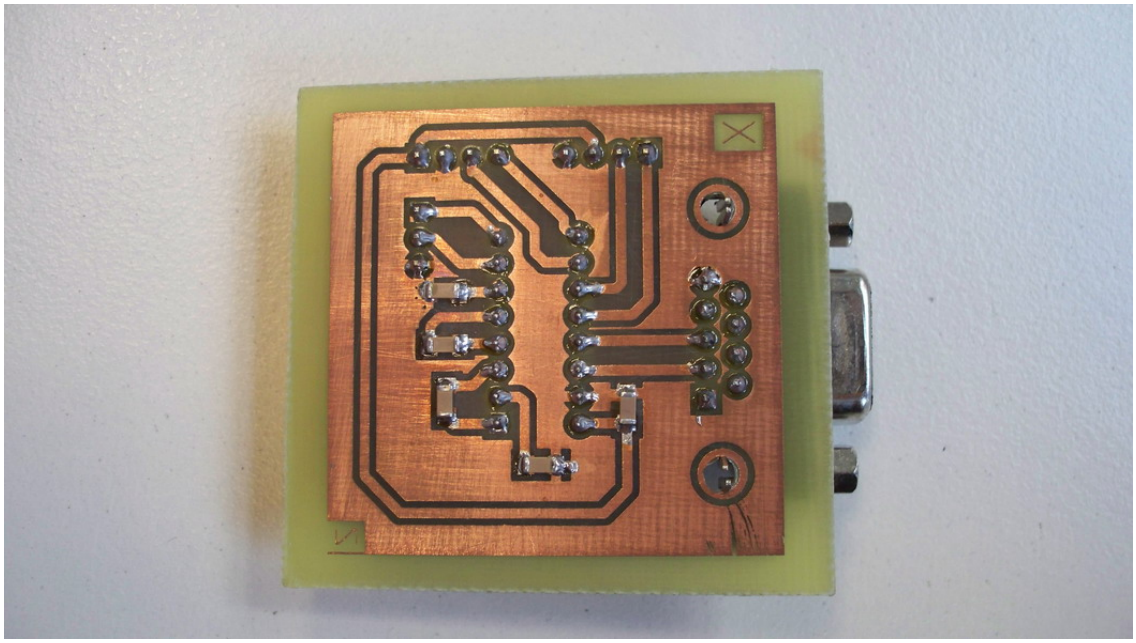
Σχήμα 5-11: Η κάτω όψη της πλακέτας του συστήματος επιταχυνσιομέτρου.



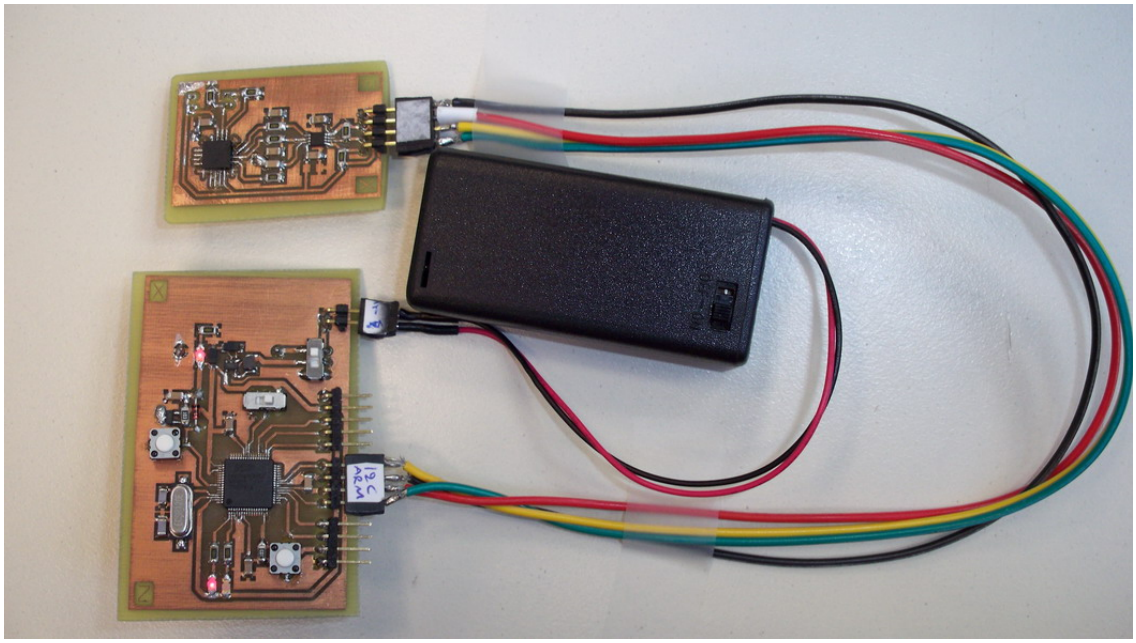
Σχήμα 5-12: Η πάνω όψη της πλακέτας του συστήματος MAX3232.



Σχήμα 5-13: Η πάνω όψη της πλακέτας του συστήματος MAX3232.



Σχήμα 5-14: Η κάτω όψη της πλακέτας του συστήματος MAX3232.



Σχήμα 5-15: Ολόκληρο το σύστημα με τις διασυνδέσεις του και εξωτερική μπαταρία.

ΚΕΦΑΛΑΙΟ 6

Εισαγωγή στον προγραμματισμό του μικροελεγκτή

6.1 Εισαγωγή

Για την δημιουργία μιας ενσωματωμένης εφαρμογής σε C κώδικα χρειάζεται κώδικας σε γλώσσα μηχανής (assembly) τον οποίο θα "τρέχει" αρχικά ο μικροελεγκτής και θα χρησιμοποιείται για την αρχικοποίησή του αλλά και θα καθορίζει κάποιες επιλογές λειτουργίας όπως το μέγεθος στοίβας (startup code). Επίσης χρειάζεται ο κώδικας εφαρμογής, η σύνδεση με κάποιες βιβλιοθήκες που χρησιμοποιούνται, αρχεία με τις διευθύνσεις μνήμης (memory map) του μικροελεγκτή και πλάνο διασύνδεσης που δινει πληροφορίες κυρίως για το μέγεθος μνήμης Flash και SRAM του μικροελεγκτή.

6.2 Προγραμματισμός σε γλώσσα μηχανής

Για να γίνει κατανοητός ο τρόπος προγραμματισμού του μικροελεγκτή θα παρουσιάσουμε απλά παραδείγματα και στην συνέχεια θα δημιουργήσουμε πιο πολύπλοκα.

Παράδειγμα 1. Βασικός κώδικας γλώσσας μηχανής (Basic startup assembler)

```
/* ex1 . s */
/* ------------------------------------------------------------
* Exception vect ors
* ------------------------------------------------------------
*/
. text
. arm
. global s t a r t
s t a r t :
/* Vectors (8 t o t a l ) */
b r e s e t /* r e s e t */
b loop /* undef ined i n s t ruc t ion */
b loop /* sof tware int e r r upt */
b loop /* p r e f e tch abort */
b loop /* data abor t */
nop /* reserved for the bootloader checksum */
b loop /* IRQ */
b loop /* FIQ */
/* ------------------------------------------------------------
```

```
* Test code
* ------------------------------------------------------------
*/
r e s e t :
l d r r0 , IODIR1
l d r r1 , IODIR1 VALUE
s t r r1 , [ r0 ]
l d r r0 , IOCLR1
s t r r1 , [ r0 ]
l d r r0 , IOSET1
l d r r1 , IOSET1 VALUE
s t r r1 , [ r0 ]
loop : b loop

/* ------------------------------------------------------------
* Constants
* ------------------------------------------------------------
*/
/* LED cont r o l r e g i s t e r s */
IOSET1 : . word 0xE0028014
IODIR1 : . word 0xE0028018
IOCLR1: . word 0xE002801C
IODIR1 VALUE: . word 0x00FF0000
IOSET1 VALUE: . word 0x00550000
. end
```

Το παραπάνω παράδειγμα ανάβει κάποια leds, που είναι συνδεδεμένα σε μερικά πινς του μικροελεγκτή. Ο μικροελεγκτής αρχίζει να βλέπει τον κώδικα που είναι μετά το _start. Εκεί βρίσκονται τα διανύσματα εξαίρεσης (exception vectors) ανά τέσσερα bytes με την εξής σειρά : επανεκκίνησης (reset), μη καθορισμένης εντολής (undefined instruction), διακοπής προγράμματος (software interrupt), ματαίωσης ανάγνωσης εντολής (prefetch abort), ματαίωσης προσπέλασης μνήμης δεδομένων (data abort), δεσμευμένος χώρος για τον bootloader, διακοπών (IRQ) και γρήγορης διακοπής (FIQ).

Δεν χρειάζεται να ρυθμίσουμε στοίβες και περιφερειακά γι' αυτό το απλό παράδειγμα. Όταν γίνεται εκκίνηση του μικροελεγκτή, αυτός διαβάζει την εντολή που υπάρχει στην θέση της εξαίρεσης reset. Αυτή είναι ένα άλμα στην θέση reset ("b reset"). Οι υπόλοιπες εξαιρέσεις δεν προβλέπεται να χρησιμοποιηθούν γι' αυτό και έχουν εντολή διακλάδωσης σε έναν ατέρμονο βρόχο ("loop: b loop").

Στο κώδικα που εκτελείται μετά το reset φορτώνεται ο καταχωρητής r0 με την διεύθυνση του καταχωρητή IODIR1 και ο r1 με την τιμή που θέλουμε να πάρει και μετά γράφουμε στην διέυθυνση που δείχνει ο r0 την τιμή του r1. Με αυτό τον τρόπο καθορίζουμε τα πινς που θα χρησιμοποιηθούν ως έξοδοι.Στην συνέχεια με την ίδια μέθοδο γράφουμε τις τιμές που θέλουμε στους καταχωρητές IOCLR1 και IOSET1.

Παρατηρούμε ότι αντί να χρησιμοποιήσουμε κατευθείαν τις διευθύνσεις των καταχωρητών, τους δώσαμε συμβολικά ονόματα (όπως IOSET1) για να κάνουμε το πρόγραμμα πιο ευανάγνωστο και για να αποφεύγονται λάθη. Όταν γράφουμε C κώδικα όλες οι διευθύνσεις (πχ. των περιφερειακών)

βρίσκονται σε ένα αρχείο. Για τον LPC2148 αυτό το αρχείο είναι το «LPC214X.h». Στην συνέχεια κάνουμε μεταγλώττιση και σύνδεση του κώδικα και παράγουμε το αρχείο σε .hex μορφή που αργότερα θα φορτωθεί στον μικροελεγκτή.

6.3 Προγραμματισμός σε γλώσσα C

Παράδειγμα 2. Πρόγραμμα σε C.

```
/* ex2 start.s */
.text
.arm
.global main
.global _start
start :
/* Vectors (8 total) */
ldr pc , main_addr /* reset */
ldr pc , loop_addr /* undefined instruction */
ldr pc , loop_addr /* software interrupt */
ldr pc , loop_addr /* prefetch abort */
ldr pc , loop_addr /* data abort */
nop /* reserved for the boot ldr checksum */
ldr pc , loop_addr /* IRQ */
ldr pc , loop_addr /* FIQ */
loop addr : .word loop
main addr : . word main
loop : b loop
. end
```

```
/* ex2 main.c */

#define IOSET1 ( *( ( volatile unsigned long *) 0xE0028014) )
#define IODIR1 ( *( ( volatile unsigned long *) 0xE0028018) )
#define IOCLR1 ( *( ( volatile unsigned long *) 0xE002801C) )

int main (void)
{
        /* Define the LED pins P1.[16..23] as output */
        IODIR1 = 0x00FF0000 ;
        /* Clear all pins */
        IOCLR1 = 0x00FF0000 ;
        /* LED[7:0] ; even on ( high ) , odd off ( low ) */
        IOSET1 = 0x00550000 ;
        while ( 1 ) ;
return 0 ;
}
```

Το παραπάνω πρόγραμμα assembly ρυθμίζει τα διανύσματα εξαίρεσης ώστε να εκτελούν έναν ατέρμονο βρόχο ενώ μόνο το διάνυσμα εξαίρεσης reset πηδάει κατευθείαν στον κώδικα C της κύριας εφαρμογής.

Τα διανύσματα εξαίρεσης καθορίζονται λίγο διαφορετικά από το παράδειγμα1 καθώς αντί να πηδάνε σε κάποια διεύθυνση, αυτή τη φορά φορτώνουν το μετρητή προγράμματος PC με την διεύθυνση της εντολής που θα εκτελέσουν.

Στον κώδικα C καθορίζονται αρχικά τα ονόματα των καταχωρητών με τις διευθύνσεις τους (πχ. IOSET1). Στο main() καθορίζονται τα πινς P1[16..23] ώς έξοδοι γράφοντας κατευθείαν την τιμή στον καταχωρητή IODIR1. Ομοίως, για να καθορίσουμε ποια πινς θα είναι high και ποια low γράφουμε τις τιμές στους IOSET1 και IOCLR1. Τέλος έχουμε μια εντολή while(1) καθώς δεν μπορεί να επιστρέψει κάπου το πρόγραμμα όταν τελειώσει όπως συμβαίνει στους ηλεκτρονικούς υπολογιστές με λειτουργικό σύστημα.

6.4 Προγραμματισμός με καθορισμό στοίβας

Όταν χρησιμοποιείται κάποια ρουτίνα στον κώδικα εφαρμογής, πρέπει να γίνει χρήση της στοίβας και συνεπώς πρέπει να καθοριστεί από που αυτή θα ξεκινάει. Στο παρακάτω πρόγραμμα γίνεται ότι και στο προηγούμενο παράδειγμα απλά τώρα γίνεται χρήση συνάρτησεων.

Παράδειγμα 3. Πρόγραμμα C με καθορισμό στοίβας.

```
/* e x3 s t a r t . s */
. text
. arm
. global main
. global s t a r t
s t a r t :
/* Vectors (8 t o t a l ) */
b r e s e t /* r e s e t */
b loop /* undef ined i n s t ruc t ion */
b loop /* sof tware int e r r upt */
b loop /* p r e f e tch abort */
b loop /* data abor t */
nop /* reserved for the bootloader checksum */
b loop /* IRQ */
b loop /* FIQ */
/* Setup the stack pointer and then jump to main */
r e s e t :
l d r sp , stack addr
bl main
/* Catch return from main */
loop : b loop
/* Constants */
/* LPC SRAM s t a r t s at 0x40000000 , and ther e i s 32Kb = 8000h */
stack addr : . word 0x40008000
. end
```

```
/* ex3a main . c */
#include " led .h"
int main (void)
{ l e d i n i t ( ) ;
led (0 x55 ) ;
```

```
while ( 1 ) ;
return 0 ;
}
```

```
/* l e d . h */
#ifndef LED H
#define LED H
/* I n i t i a l i z e the LEDs */
void l ed i n i t ( ) ;
/* Control the LEDs */
void led ( unsigned long val ) ;
/* Set LEDs */
void l e d s e t ( unsigned long s e t ) ;
/* Clear LEDs */
void l e d c l r (unsigned long c l r ) ;
#endif
```

```
/* l e d . c */
#include " led .h"
#define IOSET1 ( *( ( volatile unsigned long *) 0xE0028014) )
#define IODIR1 ( *( ( volatile unsigned long *) 0xE0028018) )
#define IOCLR1 ( *( ( volatile unsigned long *) 0xE002801C) )
void l ed i n i t ()
{ /* Define the LED pins P1 . [ 1 6 . . 2 3 ] as output */
IODIR1 = 0x00FF0000 ;
/* Clear a l l pins */
IOCLR1 = 0x00FF0000 ;
}
void led ( unsigned long val )
{ /* LEDs o f f */
IOCLR1 = (~ va l & 0xFF) << 16;
/* LEDs on */
IOSET1 = ( va l & 0xFF) << 16;
}
void l e d s e t ( unsigned long s e t )
{ IOSET1 = ( s e t & 0xFF) << 16;
}
void l e d c l r (unsigned long c l r )
{ IOCLR1 = ( c l r & 0xFF) << 16;
}
```

Το LPC2148 έχει 32kB μνήμης SRAM η οποία ξεκινάει από την διεύθυνση 40000000h και τελειώνει στην διεύθυνση 40008000h. Στους ARM η στοίβα μεγαλώνει προς τα κάτω, συνεπώς το “ex3start.s” αρχικοποιεί τον δείκτη στοίβας στο τέλος της SRAM, δηλαδή στην διεύθυνση 40008000h (“ldr sp, stack_addr” όπου stack_addr=0x4008000) και στην συνέχεια πηδάει στην main().

Το κεντρικό αρχείο C που βρίσκεται η main() συμπεριλαμβάνει το αρχείο “led.h” (“#include led.h”) καθώς η main() χρησιμοποιεί συναρτήσεις που βρίσκονται στο αρχείο “led.c” και έχουν τα πρωτότυπά τους στο “led.h”.

6.5 Καθορισμός πλάνου διασύνδεσης

Ο LPC2148 έχει 512kB μνήμης Flash και 32kB μνήμης SRAM. Οι εφαρμογές είναι συνήθως συνδεδεμένες να τρέχουν από την Flash , ενώ οι μεταβλητές που μπορούν να αλλάξουν στο πρόγραμμα και οι στοίβες χρησιμοποιούν την SRAM. Ο κώδικας που εκτελείται είναι συνδεδεμένος στο .text τμήμα ενώ μεταβλητές μόνο ανάγνωσης (read only variables) είναι συνδεδεμένες στο .rodata τμήμα. Οι αρχικοποιημένες μεταβλητές που μπορούν να μεταβληθούν συνδέονται στο .data τμήμα ενώ οι μη αρχικοποιημένες μεταβλητές στο .bss. Αν και στις εφαρμογές το .data τμήμα τρέχει από την SRAM, χρειάζεται και ένα τμήμα ίσου μεγέθους από την Flash για να αποθηκεύσει τις αρχικές τιμές που είναι καθορισμένες στην SRAM. Ο κώδικας αρχικοποίησης (startup code) αντιγράφει τις αρχικές τιμές από την Flash στην SRAM στο τμήμα .data πριν ξεκινήσει η κύρια εφραρμογή και χρησιμοποιήσει αυτές τις μεταβλητές. Ο κώδικας αρχικοποίησης πρέπει επίσης να μηδενίσει το τμήμα διευθύνσεων της SRAM που αντιστοιχεί στο .bss τμήμα. Τα παραπάνω φαίνονται στο επόμενο παράδειγμα. Επίσης χρειάζεται και ένα πλάνο διασύνδεσης το οποίο καθορίζει τον χάρτη μνήμης του επεξεργαστή, δηλαδή την τοποθεσία και το μέγεθος της Flash και SRAM, και για να καθορίσει σύμβολα για το .data τμήμα και το μέγεθος του .bss τμήματος. Το παρακάτω πλάνο διασύνδεσης χρησιμοποιείται για να μεταγλωττίσει τα παραδείγματα.

```
/* lpc2138_flash.ld
 *
 * Linker script for Philips LPC2138 ARM microcontroller
 * applications that execute from Flash.
 */

/* The LPC2138 has 512kB of Flash, and 32kB SRAM */

MEMORY
{
    flash (rx) : org = 0x00000000, len = 0x00080000
    sram  (rw) : org = 0x40000000, len = 0x00008000
}


SECTIONS
{
    /* ---------------------------------------------------------
     * .text section (executable code)
     * ---------------------------------------------------------
     */
    .text :
    {
        *start.o (.text)
        *(.text)
        *(.glue_7t) *(.glue_7)
    } > flash
    . = ALIGN(4);

    /* ---------------------------------------------------------
     * .rodata section (read-only (const) initialized variables)
     * ---------------------------------------------------------
     */
    .rodata :
    {
        *(.rodata)
    } > flash
    . = ALIGN(4);

    /* End-of-text symbols */
    _etext = . ;
    PROVIDE (etext = .);

    /* ---------------------------------------------------------
     * .data section (read/write initialized variables)
     * ---------------------------------------------------------
     *
     * The values of the initialized variables are stored
     * in Flash, and the startup code copies them to SRAM.
     *
     * The variables are stored in Flash starting at _etext,
```

```
 * and are copied to SRAM address _data to _edata.
 */
 .data : AT (_etext)
 {
      _data = . ;
      *(.data)
 } > sram
 . = ALIGN(4);

 _edata = . ;
 PROVIDE (edata = .);

/* ----------------------------------------------------------
 * .bss section (uninitialized variables)
 * ----------------------------------------------------------
 *
 * These symbols define the range of addresses in SRAM that
 * need to be zeroed.
 */
.bss :
{
    _bss = . ;
    *(.bss)
    *(COMMON)
} > sram
. = ALIGN(4);
_ebss = . ;

_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 t
    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) }
```

Ο κώδικας που εκτελείται (.text) και τα δεδομένα μόνο ανάγνωσης (.rodata) συνδέονται σε διευθύνσεις Flash. Οι αρχικοποιημένες μεταβλητές (.data) συνδέονται σε διευθύνσεις SRAM μεταξύ των συμβόλων _data και _edata, αλλά οι τιμές των των αρχικοποιημένων μεταβλητών αποθηκεύονται στην Flash μετά τα .text και .rodata τμήματα στην διεύθυνση _etext. Τα σύμβολα καθορίζονται από το αρχείο διασύνδεσης και χρησιμοποιούνται στον κώδικα αρχικοποίησης.

6.6 Κώδικας αρχικοποίησης

Ο κώδικας αρχικοποίησης καθορίζει μεταβλητές που έχουν αρχικοποιηθεί στα σύμβολα του αρχείου διασύνδεσης. Στην συνέχεια ο κώδικας αρχικοποίησης χρησιμοποιεί τα σύμβολα του τμήματος δεδομένων για να αντιγράψει τις τιμές από την Flash στην SRAM. Η εφαρμογή χρησιμοποιεί τις μεταβλητές της μνήμης SRAM. Το .bss τμήμα περιέχει μη αρχικοποιημένα δεδομένα. Το αρχείο διασύνδεσης συνδυάζει τα διάφορα .bss τμήματα από διάφορα αρχεία αντικειμένων που αποτελούν μια εφαρμογή και καθορίζει την αρχική (_bss) και την τελική (_ebss) διεύθυνση για τις μη αρχικοποιημένες μεταβλητές. Ο κώδικας αρχικοποίησης πρέπει να μηδενίσει την SRAM μεταξύ των _bss και _ebss. Ο κώδικας αρχικοποίησης που χρησιμοποιούμε στα παραδείγματα είναι:

```
/*ex4start.s*/
.global main
.global start
/*Symbols defined by the link er script*/
.global etext
.global data
.global edata
.global bss
.global ebss
.text
.arm
start:
/*Vectors (8 total)*/
breset/*reset*/
```

```
b loop /* undef ined i n s t ruc t ion */
b loop /* sof tware int e r r upt */
b loop /* p r e f e tch abort */
b loop /* data abor t */
nop /* reserved for the bootloader checksum */
b loop /* IRQ */
b loop /* FIQ */

/* Setup C runtime :
* - copy . data s e c t ion to SRAM
* - c l e a r . bss
* - setup stack pointer
* - jump to main
*/
r e s e t :
/* Copy . data */
l d r r0 , data source
l d r r1 , data s ta r t
l d r r2 , data end
copy data :
cmp r1 , r2
ldrne r3 , [ r0 ] , #4
s trne r3 , [ r1 ] , #4
bne copy data
/* Clear . bs s */
l d r r0 , =0
l d r r1 , bs s s t a r t
l d r r2 , bss end
c l e a r b s s :
cmp r1 , r2
s trne r0 , [ r1 ] , #4
bne c l e a r b s s
/* Stack pointer */
l d r sp , stack addr
bl main
/* Catch return from main */
loop : b loop
/* Constants */
/* LPC SRAM s t a r t s at 0x40000000 , and ther e i s 32Kb = 8000h */
stack addr : . word 0x40008000
/* Linker symbols */
data source : . word etext
data s ta r t : .word data
data end : . word edata
b s s s t a r t : .word bss
bss end : . word ebss
. end
```

Ο κύριος κώδικας χρησιμοποιεί και καθυστερήσεις για να αναβοσβύνει leds :

```
/* ex4b main . c */
#include " led .h"
static int l e d value [ 2 ] = {0x55 , 0xAA} ;
int main (void)
{
```

```
 int i ;
l e d i n i t ( ) ;
while (1) { led ( led value [ 0 ] ) ;
for ( i = 0; i < 0x50000 ; i++);
led ( led value [ 1 ] ) ;
for ( i = 0; i < 0x50000 ; i++);
}
return 0 ;
}
```

6.7 Αρχικοποίηση επεξεργαστή

6.7.1 Γενικά

Οι μικροελεγκτές LPC21xx απαιτούν μια συγκεκριμένη διαδικασία αρχικοποίησης η οποία αποτελείται από τα παρακάτω βήματα:

1) Ρύθμιση διανυσμάτων εξαίρεσης
2) Ρύθμιση PLL
3) Ρύθμιση MAM
4) Καθορισμός δεικτών στοίβας για κάθε κατάσταση λειτουργίας του επεξεργαστή
5) Αντιγραφή .data τμήματος στην SRAM
6) Καθαρισμός .bss τμήματος
7) Άλμα στην main()

Η αρχικοποίηση ξεκινάει από τον καθορισμό των διανυσμάτων εξαίρεσης αφού αυτά τα διανύσματα ξεκινάνε από την διεύθυνση μηδέν. Στη συνέχεια το PLL και η MAM ρυθμίζονται για να τρέχει ο κώδικας στην επιθυμητή ταχύτητα.

6.7.2 Ρύθμιση PLL

Αν το PLL ρυθμιστεί πριν την ρύθμιση των δεικτών στοίβας, η αρχικοποίησή του πρέπει να γίνει σε γλώσσα μηχανής. Ένας διαφορετικός τρόπος είναι να ρυθμιστούν πρώτα οι στοίβες και μετά να καλέσουμε μια συνάρτηση σε C πριν γίνει άλμα στην main(). Ο κώδικας σε C για το PLL είναι :

```
#define PLLCON (*(volatile unsigned int *)0xE01FC080)
#define PLLCFG (*(volatile unsigned int *)0xE01FC084)
#define PLLSTAT (*(volatile unsigned int *)0xE01FC088)
#define PLLFEED (*(volatile unsigned int *)0xE01FC08C)
#define PLLCON_PLLE (1 << 0)
#define PLLCON_PLLC (1 << 1)
#define PLLSTAT_PLOCK (1 << 10)
#define PLLFEED1 0xAA
#define PLLFEED2 0x55
#define PLLCFG_VALUE 0x24

void pll_init(void)
{
PLLCFG = PLLCFG_VALUE;
PLLCON = PLLCON_PLLE;
```

```
PLLFEED = PLLFEED1;
PLLFEED = PLLFEED2;
while ((PLLSTAT & PLLSTAT_PLOCK) == 0);
PLLCON = PLLCON_PLLC|PLLCON_PLLE;
PLLFEED = PLLFEED1;
PLLFEED = PLLFEED2;
}
```

Η ρύθμιση του PLL σε γλώσσα μηχανής είναι:

```
/* Constants (and storage, used in ldr statements) */
PLLBASE: .word 0xE01FC080
/* Constants (used as immediate values) */
.equ PLLCON_OFFSET, 0x0
.equ PLLCFG_OFFSET, 0x4
.equ PLLSTAT_OFFSET, 0x8
.equ PLLFEED_OFFSET, 0xC
.equ PLLCON_PLLE, (1 << 0)
.equ PLLCON_PLLC, (1 << 1)
.equ PLLSTAT_PLOCK, (1 << 10)
.equ PLLFEED1, 0xAA
.equ PLLFEED2, 0x55
.equ PLLCFG_VALUE, 0x24
pll_init:
/* Use r0 for indirect addressing */
ldr r0, PLLBASE
/* PLLCFG = PLLCFG_VALUE */
mov r3, #PLLCFG_VALUE
str r3, [r0, #PLLCFG_OFFSET]
/* PLLCON = PLLCON_PLLE */
mov r3, #PLLCON_PLLE
str r3, [r0, #PLLCON_OFFSET]
/* PLLFEED = PLLFEED1, PLLFEED2 */
mov r1, #PLLFEED1
mov r2, #PLLFEED2
str r1, [r0, #PLLFEED_OFFSET]
str r2, [r0, #PLLFEED_OFFSET]
/* while ((PLLSTAT & PLLSTAT_PLOCK) == 0); */
pll_loop:
ldr r3, [r0, #PLLSTAT_OFFSET]
tst r3, #PLLSTAT_PLOCK
beq pll_loop
/* PLLCON = PLLCON_PLLC|PLLCON_PLLE */
mov r3, #PLLCON_PLLC|PLLCON_PLLE
str r3, [r0, #PLLCON_OFFSET]
/* PLLFEED = PLLFEED1, PLLFEED2 */
str r1, [r0, #PLLFEED_OFFSET]
str r2, [r0, #PLLFEED_OFFSET]
```

6.7.3 Ρύθμιση MAM

Η ρύθμιση της MAM σε γλώσσα μηχανής είναι:

```
/* Constants (and storage, used in ldr statements) */
MAMBASE: .word 0xE01FC000
/* Constants (used as immediate values) */
.equ MAMCR_OFFSET, 0x0
.equ MAMTIM_OFFSET, 0x4
.equ MAMCR_VALUE, 0x2 /* fully enabled */
.equ MAMTIM_VALUE, 0x4 /* fetch cycles */
mam_init:
/* Use r0 for indirect addressing */
ldr r0, MAMBASE
/* MAMCR = MAMCR_VALUE */
mov r1, #MAMCR_VALUE
str r1, [r0, #MAMCR_OFFSET]
/* MAMTIM = MAMTIM_VALUE */
mov r1, #MAMTIM_VALUE
str r1, [r0, #MAMTIM_OFFSET]
```

6.7.4 Ρύθμιση Στοιβών

Οι ARM έχουν επτά καταστάσεις λειτουργίας και έξι διαφορετικούς δείκτες στοίβας. Αυτές είναι χρήστη/συστήματος, επιβλέπων, IRQ, FIQ, ματαίωσης και απροσδιόριστης κατάστασης. Ο επεξεργαστής ARM αρχίζει σε κατάσταση επίβλεψης η οποία είναι μια προνομιακή κατάσταση. Οι καταχωρητές ελέγχου και κατάστασης προγράμματος CPSR μπορούν να ρυθμιστούν μέσω μιας προνομιακής κατάστασης για να αλλάξουν την κατάσταση λειτουργίας του επεξεργαστή και για να καθορίστούν οι διάφορες στοίβες.

Το μέγεθος της στοίβας για κάθε κατάσταση λειτουργίας εξαρτάται από την εφαρμογή. Όταν μια εξαίρεση δημιουργηθεί, οι καταχωρητές διασύνδεσης LR και οι καταχωρητές κατάστασης SPSR για την κατάσταση εξαίρεσης, χρησιμοποιούνται για να σώσουν την κατάσταση του επεξεργαστή. Ο επεξεργαστής ARM δεν χρησιμοποιεί την στοίβα κατά την είσοδο στην εξαίρεση, αλλά είναι μόνο ο κώδικας χειρισμού της εξαίρεσης που χρησιμοποιεί την στοίβα. Οι στοίβες που χρειάζονται γενικά είναι αυτές της κατάστασης συστήματος και επίβλεψης για την λειτουργία λειτουργικού συστήματος, στοίβα χρήστη για χρήση εφαρμογών, οι IRQ και FIQ στοίβες για τις ρουτίνες εξυπηρέτησης διακοπών και προαιρετικά οι στοίβες ματαίωσης και απροσδιόριστης κατάστασης. Ένα παράδειγμα για την αρχικοποίηση των στοιβών είναι:

```
/* Constants (and storage, used in ldr statements) */
STACK_START: .word 0x40008000
/* Constants (used as immediate values) */
/* Processor modes (see pA2-11 ARM-ARM) */
.equ FIQ_MODE, 0x11
.equ IRQ_MODE, 0x12
.equ SVC_MODE, 0x13 /* reset mode */
.equ ABT_MODE, 0x17
.equ UND_MODE, 0x1B
```

```
.equ SYS_MODE, 0x1F
/* Stack sizes */
.equ FIQ_STACK_SIZE, 0x00000080 /* 32x32-bit words */
.equ IRQ_STACK_SIZE, 0x00000080
.equ SVC_STACK_SIZE, 0x00000080
.equ ABT_STACK_SIZE, 0x00000010 /* 4x32-bit words */
.equ UND_STACK_SIZE, 0x00000010
.equ SYS_STACK_SIZE, 0x00000400 /* 256x32-bit words */
/* CPSR interrupt disable bits */
.equ IRQ_DISABLE, (1 << 7)
.equ FIQ_DISABLE, (1 << 6)
/* Setup the stacks */
ldr r0, STACK_START
/* FIQ mode stack */
msr CPSR_c, #FIQ_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
sub r0, r0, #FIQ_STACK_SIZE
/* IRQ mode stack */
msr CPSR_c, #IRQ_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
sub r0, r0, #IRQ_STACK_SIZE
/* Supervisor mode stack */
msr CPSR_c, #SVC_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
sub r0, r0, #SVC_STACK_SIZE
/* Undefined mode stack */
msr CPSR_c, #UND_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
sub r0, r0, #UND_STACK_SIZE
/* Abort mode stack */
msr CPSR_c, #ABT_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
sub r0, r0, #ABT_STACK_SIZE
/* System mode stack */
msr CPSR_c, #SYS_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
/* Leave the processor in system mode */
```

Ο κώδικας αρχικοποίησης θέτει τελευταία την στοίβα συστήματος και αφήνει τον επεξεργαστή σε κατάσταση συστήματος. Ο επεξεργαστής δεν αφήνεται σε κατάσταση επίβλεψης γιατί μια διακοπή προγράμματος (SWI) τον οδηγεί σε κατάσταση επίβλεψης. Οι διακοπές δεν πρέπει να είναι ενεργοποιημένες όταν ο επεξεργαστής είναι σε κατάσταση εξαίρεσης γιατί το περιεχόμενο του καταχωρητή διασύνδεσης LR μπορεί να χαθεί.

6.8 Διαχείρηση εξαιρέσεων

Οι επξεξεργαστές ARM έχουν επτά καταστάσεις λειτουργίας και πέντε καταστάσεις εξαίρεσης. Αυτές είναι:

1) Γρήγορη διακοπή (FIQ)
2) Κανονική διακοπή (IRQ)
3) Ματαιώσεις μνήμης, οι οποίες μπορούν να χρησιμοποιηθούν για να υλοποιήσουμε προστασία μνήμης και εικονική μνήμη
4) Προσπάθεια εκτέλεσης μιας απροσδιόριστης εντολής

5) Διακοπή προγράμματος (SWI), η οποία μπορεί να χρησιμοποιηθεί για να γίνει κλήση σε λειτουργικό σύστημα

Κάθε κατάσταση εξαίρεσης έχει δικούς του δείκτες στοίβας SP (R13) και καταχωρητές διασύνδεσης LR (R14). Η εξαίρεση γρήγορης διακοπής FIQ έχει επιπλέον καταχωρητές (R8-R12) για να μειώσει τον χρόνο αποθήκευσης και επαναφοράς των καταχωρητών αυτών. Όταν συμβεί μια εξαίρεση ο καταχωρητής διασύνδεσης LR κρατάει την διεύθυνση επιστροφής για την επεξεργασία εξαίρεσης. Ο SPSR σώζει την κατάσταση του CPSR την στιγμή της εξαίρεσης. Η προτεινόμενη διαδικασία εισόδου και εξόδου για μια διακοπή είναι:

```
sub lr, lr, #4
stmfd sp!, {<other_registers>, lr}

... interrupt handler ...

ldmfd sp!, {<other_registers>, pc}^
```

Ένας χειριστής εξαίρεσης (exception handler) μπορεί να γραφτεί σε γλώσσα μηχανής, ή μπορούν να χρησιμοποιηθούν συγκεκριμένες C λέξεις κλειδιά για να δημιουργήσουν τον κατάλληλο κώδικα εισόδου και εξόδου. Ο μεταγλωττιστής GCC έχει συγκεκριμένες non-ANSI λέξεις επέκτασης για την δήλωση χειριστών εξαίρεσης σε C κώδικα. Η δήλωση ενός χειριστή για μια IRQ εξαίρεση είναι :

```
void irq_handler(void) __attribute__ ((interrupt("IRQ")));
```

Οι λέξεις κλειδιά πηγών εξαίρεσης είναι IRQ, FIQ, SWI, ABORT και UNDEF. Ο κώδικας αρχικοποίησης, όταν εχουμε FIQ και IRQ διακοπές, πρέπει να καθοριστεί ώστε τα διανύσματα διακοπών να βρίσκονται στις κατάλληλες διευθύνσεις.

```
_start:
b reset /* reset */
b loop /* undefined instruction */
b loop /* software interrupt */
b loop /* prefetch abort */
b loop /* data abort */
nop /* reserved for the bootloader checksum */
ldr pc, [pc, #-0x0FF0] /* VicVectAddr */
ldr pc, fiq_addr

/* Address of the handler function */
fiq_addr: .word fiq_handler
```

Παρατηρούμε ότι το διάνυσμα IRQ, φορτώνει τον μετρητή προγράμματος με την διεύθυνση του καταχωρητή διευθύνσεων διανυσμάτων VicVectAddr του VIC. Αυτό σημαίνει ότι θα εκτελεστεί η ρουτίνα εξυπηρέτησης διακοπής που δείχνει ο VIC. Το διάνυσμα FIQ φορτώνει στον μετρητή προγράμματος την διεύθυνση του χειριστή FIQ (fiq_handler) ώστε όταν έχουμε FIQ διακοπή να εκτελεστεί η ρουτίνα που δείχνει αυτή η διεύθυνση.

ΚΕΦΑΛΑΙΟ 7

Κώδικας εφαρμογής

7.1 Γενικά

Ο κώδικας εφαρμογής γράφτηκε στο Programmer's Notepad και μεταγλωττίστηκε με το WinARM. Τα αρχεία που αποτελούν ολόκληρη την εφαρμογή είναι τα εξής:

1) main.c
2) lpc214x.h
3) armlibdefs.h
4) armlibtypes.h
5) board.h
6) delays.c
7) delays.h
8) i2c.c
9) i2c.h
10) max1237.c
11) max1237.h
12) uart.c
13) uart.h
14) rprintf.c
15) rprintf.h
16) includes.h
17) boot.s
18) Makefile
19) lpc2138-rom.ld

Το αρχείο main.c είναι το κεντρικό αρχείο της εφαρμογής. Ο κώδικας που υπάρχει σε αυτό εκτελείται από τον μικροελεγκτή. Επίσης χρησιμοποιεί συναρτήσεις και δεδομένα που ορίζονται σε άλλα αρχεία. Το armlibdefs.h περιέχει χρήσιμες μακροεντολές που κάνουν το πρόγραμμα πιο ευανάγνωστο κυρίως παρέχοντας μακροεντολές για χειρισμό των bits. Το armlibtypes.h καθορίζει εύκολα μνημονικά ονόματα για τους τύπους δεδομένων που χρησιμοποιούνται. Το board.h καθορίζει σε ποια πινς του μικροελεγκτή είναι συνδεδεμένα leds και κουμπιά και παρέχει μακροεντολές για τον χειρισμό τους. Τα αρχεία delays.h και delays.c έχουν μια συνάρτηση που παράγει καθυστέρηση. Τα i2c.h και i2c.c χρησιμοποιούνται για την επικοινωνία με το πρωτόκολλο και το περιφερειακό i2c του μικροελεγκτή. Τα max1237.h και max1237.c επιτρέπουν την επικοινωνία με το ολοκληρωμένο max1237. Τα uart.h και uart.c έχουν συναρτήσεις για την σειριακή επικοινωνία και τα rprintf

αρχεία παρέχουν μια πιο ολοκληρωμένη συναρτηση για την σειριακή θύρα. Το includes.h περιέχει όλα τα header (“.h”) αρχεία που χρησιμοποιούνται ώστε να είναι όλα μαζεμένα. Το boot.s περιέχει τον κώδικα αρχικοποίησης σε γλώσσα μηχανής και το lpc2138-rom.ld είναι το πλάνο διασύνδεσης για τον μικροελεγκτή. Τέλος το Makefile είναι ένα αρχείο που παρέχει πληροφορίες στον μεταγλωττιστή για τον τρόπο που θα γίνει η μεταγλώττιση του κώδικα εφαρμογής.

7.2 Το αρχείο “main.c”

Το αρχείο main.c είναι το κεντρικό αρχείο του κώδικα εφαρμογής. Μετά τον κώδικα αρχικοποίησης boot.s ο μικροελεγκτής αρχίζει να εκτελεί το main.c. Αυτό είναι :

```
//#############################################################################################
//############################## ARM ACCELEROMETER ############################################
//#############################################################################################

/* includes */
#include <stdlib.h>
#include "includes.h"

#define NSAMPLES 5000
#define T100NSEC 50                             //T=T100NSEC*100nsec

u08 adc_data[10];
u16 acc_data[3];
u16 data[3][NSAMPLES];
volatile u32 timer1_counter;

/* General purpose timers */
volatile u32        timer_1sec;
volatile u32        timer_10ms;

// Configures the processor
static void SetupHardware( void );

// Timer 1 functions
void irq_timer1_handler(void) __attribute__ ((interrupt("IRQ")));
void init_timer1(void);
void stop_timer1(void);
void start_timer1(void);

//#############################################################################################
//################################## MAIN #####################################################
//#############################################################################################

int main( void )
{
        /* Setup the hardware */
        SetupHardware();
        init_UART0(9600);
        rprintf_devopen( sendc_UART0 );
        rprintf("\nrprintf: %s", "rprintf init ok.\n");
        IODIR0 = LED_1 | LED_2 ;

        timer1_counter =0;
        init_timer1();
        stop_timer1();
        asm volatile ("msr CPSR_c,#0x13 | 0x40");    //enable IRQ's

        i2cInit();
```

```
	max1237_init();

	LEDS_OFF;

	while(1)
	{
			while(!BUTTON_1_PRESSED);	//perimenei na patithei to button
			delay(20000);
			while(BUTTON_1_PRESSED);	//perimenei na afisoume to button
			delay(10000000);
			LED_1_ON;			//anavei to LED_1
			rprintf("Button pressed\n");

			for(u32 i=0;i<NSAMPLES;i++)	//pairnei ta samples
			{
				if(max1237_read(adc_data)==I2C_OK)
				{
					acc_data[0]= (0x0f&adc_data[0])*256 + adc_data[1];
					acc_data[1]= (0x0f&adc_data[2])*256 + adc_data[3];
					acc_data[2]= (0x0f&adc_data[4])*256 + adc_data[5];

					data[0][i] = acc_data[0];
					data[1][i] = acc_data[1];
					data[2][i] = acc_data[2];
				}
				timer1_counter = 0;
				start_timer1();
				do{
					;
				}while(timer1_counter!= T100NSEC);
				stop_timer1();
			}

			rprintf("samples collected\n");
			LED_2_ON;			//anavei kai to LED_2
			while(!BUTTON_1_PRESSED);	//perimenei na patithei to button
			delay(20000);
			while(BUTTON_1_PRESSED);	//perimenei na afisoume to button
			delay(10000000);
			LED_1_OFF;			//svinei to LED_1
			rprintf("button pressed again\n");

			for(u32 j=0;j<NSAMPLES;j++)	//stelnei ta samples
			{
				rprintf("%d %d %d %d\n",j,data[0][j],data[1][j],data[2][j]);
				delay(50000);
				if(j%1000 == 0)
					delay(500000);
			}
			LED_2_OFF;	//svinei kai to LED_2
			rprintf("data sent\n");
	}  //end while(1)

	while(1);
	return 0;
}
//#############################################################################

static void SetupHardware( void )
{
	/* Configure the RS2332 pins.  All other pins remain at their default of 0. */
	PINSEL0 |= 0x0001;
	PINSEL0 |= 0x0004;

	/* Setup the PLL to multiply the XTAL input by 4. */
	PLLCFG = 4 | (0<<6) | (1<<5);
```

```
	/* Activate the PLL by turning it on then feeding the correct sequence of bytes. */
	PLLCON = 0x0001;
	PLLFEED = 0xaa;
	PLLFEED = 0x55 ;

	/* Wait for the PLL to lock... */
	while( !( PLLSTAT & 0x0400 ) );

	/* ...before connecting it using the feed sequence again. */
	PLLCON = 0x0003;
	PLLFEED = 0xaa;
	PLLFEED = 0x55 ;

	/* Setup and turn on the MAM.  Three cycle access is used due to the fast
	PLL used.  It is possible faster overall performance could be obtained by
	tuning the MAM and PLL settings. */
	MAMTIM = 0x03;
	MAMCR = 0x02;

	/* Setup the peripheral bus to be the same as the PLL output. */
	VPBDIV =  0x01;
}
//#############################################################################################
void init_timer1(void)
{
	T1TCR = BIT(1);		//counter disabled and reset
	T1CTCR = 0x00;		//timer mode : every PCLK edge
	T1PR = 0;			//counter counts every PCLK edge
	T1PC = 0;
	T1MR0 = 6000;		// 100nsec
	T1MCR = BIT(0) | BIT(1) ; //interrupt and TC reset when match
	T1TCR = BIT(0);  //counter1 enable

	VICIntSelect &= ~BIT(5) ; //timer1 Vectored IRQ
	VICVectCntl1 = 0x20 | 5 ;
	VICVectAddr1 = (DWORD)irq_timer1_handler;
	VICIntEnable = BIT(5) ;
}
//#############################################################################################
void irq_timer1_handler (void)
{
	timer1_counter++;
	T1IR = BIT(0);//T1IR;		// Clear interrupt flag
	VICVectAddr = 0;		// Acknowledge Interrupt
}
//#############################################################################################
void stop_timer1(void)
{
	VICIntEnClr = BIT(5); //disable timer1 interrupt
	T1TCR = BIT(1);		// disable timer1 counting
}
//#############################################################################################
void start_timer1(void)
{
	VICIntEnable = BIT(5) ;
	T1TCR = BIT(0) | BIT(1) ;  //timer1 reset and enabled
	T1TCR = BIT(0);
}
```

Το αρχείο main.c ξεκινάει με τα αρχείο που συμπεριλαβάνει τις συναρτήσεις και τις μακροεντολές που χρησιμοποιούνται από το πρόγραμμα. Στην συνέχεια ορίζονται οι σταθερές που καθορίζουν τον αριθμό δειγμάτων και την καθυστέρηση για την περίοδο δειγματοληψίας. Έπειτα ορίζονται οι μεταβλητές

του προγράμματος και τα πρότυπα των συναρτήσεων που ορίζονται μέσα στο main.c.

Αρχικά μέσα στην συνάρτηση main() καλείται η συνάρτηση SetupHardware(). Αυτή όπως μπορούμε να δούμε καθορίζει δύο πινς του μικροελεγκτή για να χρησιμοποιηθούν για την σειριακή επικοινωνία. Στην συνέχεια ρυθμίζει το PLL με τον τρόπο που έχουμε περιγράψει ώστε να πολλαπλασιάζει την συχνότητα του εξωτερικού κρυστάλλου των 12MHz με το πέντε ώστε η ταχύτητα λειτουργίας του μικροελεγκτή να είναι 60MHz. Έπειτα καθορίζουμε την μονάδα MAM ώστε να γίνεται γρήγορη πρόσβαση στην μνήμη Flash. Τέλος καθορίζεται η συχνότητα λειτουργίας των περιφερειακών του μικροελεγκτή ίση με την συχνότητα του επεξεργαστή θέτοντας την τιμή του διαιρέτη ίση με ένα (VPBDIV=1).

Μετά την SetupHardware() καλείται η ρουτίνα αρχικοποίησης του UART0. Η init_UART0() δέχεται σαν όρισμα την τιμή ταχύτητας μεταφοράς δεδομένων. Στην συγκεκριμένη περίπτωση είναι 9600 δηλαδή η σειριακή επικοινωνία γίνεται με ταχύτητα 9600kbps. Η rprintf_devopen() ορίζει ποιον τρόπο αποστολής δεδομένων θα χρησιμοποιεί η rprintf() για να στέλνει δεδομένα. Αυτός ορίζεται με το όρισμα sendc_UART0 και έπειται καλείται η rprintf() για να τυπώσει ένα μήνυμα ότι αρχικοποιήθηκε η σειριακή επικοινωνία και η rprintf. Η εντολή "IODIR0 = LED_1 | LED_2 ;" ορίζει τα πινς που είναι συνδεδεμένα τα δύο LEDS να χρησιμοποιούνται σαν έξοδοι.

Μετά γίνεται η αρχικοποίηση του περιφερειακού χρονομετρητή timer1 και ενεργοποιούνται οι διακοπές IRQ. Η αρχικοποίηση γίνεται με τη συνάρτηση init_timer1(). Σε αυτή ρυθμίζεται ο χρονομετρητής για να αυξάνει την τιμή του κατά ένα με κάθε παλμό του ρολογιού των 60MHz. Όταν η τιμή του φτάσει την τιμή 6000 (δηλαδή κάθε 100nsec) που έχει ο καταχωρητής συγκρισης δημιουργείται διακοπή και μηδενίζεται ο timer1. Η διακοπή είναι διανυσματική στη θέση ένα. Στη ρουτίνα εξυπηρέτησης διακοπής "irq_timer1_handler()" αυξάνεται η τιμή της timer1_counter μεταβλητής κατά ένα. Αυτή την μεταβλητή την χρησιμοποιεί το κυρίως πρόγραμμα για να δημιουργεί καθυστερήσεις ακριβείας σε πολλαπλάσια των 100nsec.

Για την ενεργοποίηση των διακοπών χρησιμοποιείται η εντολή σε γλώσσα μηχανής "asm volatile ("msr CPSR_c,#0x13 | 0x40");" καθώς δεν επιτρέπεται η προσπέλαση στον CPSR στην γλώσσα C.

Η συνάρτηση i2cInit() αρχικοποιεί και ενεργοποιεί το περιφερειακό I2C του μικροελεγκτή. Μετά από αυτή την συνάρτηση μπορεί να χρησιμοποιηθεί ο διάδρομος I2C και έτσι καλείται η συνάρτηση max1237_init() η οποία αρχικοποιεί το ολοκληρωμένο max1237 και το προετοιμάζει για επικοινωνία μαζί του. Μόλις γίνει αυτό σβύνουν τα leds της πλακέτας και ο μικροελεγκτής περιμένει να πατηθεί το κουμπί. Μόλις το πατήσουμε και το αφήσουμε δημιουργείται μια καθυστέρηση, ανάβει το led1 και τυπώνει ένα μήνυμα με την rprintf(). Έπειτα ο μικροελεγκτής διαβάζει NSAMPLES δείγματα καλώντας την συνάρτηση max1237_read(). Κάθε φορά που την καλεί καθυστερεί και 5msec (T100NSEC*100nsec) ώστε η συχνότητα δειγματοληψίας να είναι περίπου 200Hz που είναι αρκετή για την κίνηση του ανθρώπου.

Μόλις ληφθούν όλα τα δείγματα τυπώνεται ένα μήνυμα, ανάβει το led2 και περιμένουμε να πατηθεί το κουμπί. Αφού γίνει αυτό σβύνει το led1 και στέλνονται τα δεδομένα μέσω της σειριακής θύρας. Όταν ολοκληρωθεί η αποστολή, σβύνει το led2 και η διαδικασία μπορεί να επαναληφθεί ξανά.

7.3 Το αρχείο “LPC214x.h”

Το αρχείο LPC214x.h είναι ένα αρχείο επικεφαλίδας που περιλαμβάνει όλες τις ονομασίες των καταχωρητών που υπάρχουν στον μικροελεγκτή LPC2148 και τις διευθύνσεις τους. Στην ουσία καθορίζει ονομαστικές τιμές στις τιμές διευθύνσεων των καταχωρητών ώστε μέσα από το πρόγραμμά μας να χρησιμοποιούμε τις μνημονικές αυτές τιμές που κάνουν το πρόγραμμα πιο ευανάγνωστο, επιτρέπει πιο γρήγορη ανάπτυξη κώδικα και με λιγότερα λάθη. Ένα απόσπασμα από αυτό το αρχείο παρουσιάζεται παρακάτω:

```
/* Pin Connect Block */
#define PINSEL_BASE_ADDR        0xE002C000
#define PINSEL0       (*(volatile unsigned long *)(PINSEL_BASE_ADDR + 0x00))
#define PINSEL1       (*(volatile unsigned long *)(PINSEL_BASE_ADDR + 0x04))
#define PINSEL2       (*(volatile unsigned long *)(PINSEL_BASE_ADDR + 0x14))

/* General Purpose Input/Output (GPIO) */
#define GPIO_BASE_ADDR          0xE0028000
#define IOPIN0        (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x00))
#define IOSET0         (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x04))
#define IODIR0        (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x08))
#define IOCLR0         (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x0C))
#define IOPIN1        (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x10))
#define IOSET1         (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x14))
#define IODIR1        (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x18))
#define IOCLR1         (*(volatile unsigned long *)(GPIO_BASE_ADDR + 0x1C))
```

Στον παραπάνω κώδικα παρουσιάζονται οι καταχωρητές που σχετίζονται με το Pin Select Block και με τις εισόδους/εξόδους γενικής χρήσης GPIO. Παρατηρούμε ότι οι αρχικά καθορίζεται μια γενική βάση διευθύνσεων που σχετίζεται με το περιφερειακό και στην συνέχεια για κάθε καταχωρητή χρησιμοποιείται η διεύθυνση βάσης συν μια άλλη τιμή. Οι τιμές διευθύνσεων των καταχωρητών δίνονται από τον κατασκευαστή και παρουσιάζονται στο datasheet του μικροελεγκτή. Στον κώδικα εφαρμογής μπορούμε να χρησιμοποιούμε το όνομα του καταχωρητή που έχει καθοριστεί (πχ. IODIR0) και όχι την διεύθυνση του (πχ. 0xE0028008). Τέλος, τους δίνονται τιμές volatile, διαφορετικά ο μεταγλωττιστής θα έκανε λάθη κατά τη βελτιστοποίηση του προγράμματος.

7.4 Το αρχείο “armlibdefs.h”

Το αρχείο armlibdefs.h είναι αρχείο επικεφαλίδας που σε αυτό ορίζονται ορισμένες χρήσιμες μακροεντολές.

```
#ifndef ARMLIBDEFS_H
#define ARMLIBDEFS_H

// MIN/MAX/ABS macros
#define MIN_VAL(a,b)                    (((a)<(b))?(a):(b))
#define MAX_VAL(a,b)                    (((a)>(b))?(a):(b))
#define ABS(x)                          (((x)>0)?(x):(-x))

// constants
#define PI          3.14159265359

// some handy defines
#ifndef BIT
#define BIT(n)                          (1 << (n))
#endif
#ifndef BV
#define BV(n)                           (1 << (n))
#endif

#define sbi(num,bitloc)             (num|=(1<<bitloc))
#define cbi(num,bitloc)             (num&=~(1<<bitloc))
#define gbi(num,bitloc)             ((num&=(1<<bitloc))>>bitloc)

// use this for packed structures
#define GNUC_PACKED __attribute__((packed))
// port bits
#define P00                 BIT(0)      // P0.00
#define P01                 BIT(1)      // P0.01
#define P02                 BIT(2)      // P0.02
#define P03                 BIT(3)      // P0.03
#define P04                 BIT(4)      // P0.04
#define P05                 BIT(5)      // P0.05
#define P06                 BIT(6)      // P0.06
#define P07                 BIT(7)      // P0.07
#define P08                 BIT(8)      // P0.08
#define P09                 BIT(9)      // P0.09
#define P10                 BIT(10)         // P0.10
#define P11                 BIT(11)         // P0.11
#define P12                 BIT(12)         // P0.12
#define P13                 BIT(13)         // P0.13
#define P14                 BIT(14)         // P0.14
#define P15                 BIT(15)         // P0.15
#define P16                 BIT(16)         // P0.16
#define P17                 BIT(17)         // P0.17
#define P18                 BIT(18)         // P0.18
#define P19                 BIT(19)         // P0.19
#define P20                 BIT(20)         // P0.20
#define P21                 BIT(21)         // P0.21
#define P22                 BIT(22)         // P0.22
#define P23                 BIT(23)         // P0.23
```

```
#define P24                BIT(24)          // P0.24
#define P25                BIT(25)          // P0.25
#define P26                BIT(26)          // P0.26
#define P27                BIT(27)          // P0.27
#define P28                BIT(28)          // P0.28
#define P29                BIT(29)          // P0.29
#define P30                BIT(30)          // P0.30
#define P31                BIT(31)          // P0.31

#endif
```

Αρχικά ορίζονται οι μακροεντολές MIN_VAL(), MAX_VAL() και ABS() οι οποίες υπολογίζουν την μικρότερη και την μεγαλύτερη από δύο τιμές και την απόλυτη τιμή ενός αριθμού αντίστοιχα.

Επίσης ορίζεται η πολύ χρήσιμη μακροεντολή BIT(n) η οποία δίνει τον αριθμό που έχει στην θέση n την μονάδα. Αυτή η μακροεντολή χρησιμοποιείται κυρίως για να θέτει συγκεκριμένα bits σε έναν καταχωρητή. Για παράδειγμα αν θέλουμε να θέσουμε τα bits 3 και 7 ενός καταχωρητή reg γράφουμε στον κώδικα reg | = BIT(3) | BIT(7). Επίσης αν θέλουμε να μηδενίσουμε το bit 4 του reg γράφουμε reg &= ~BIT(4). Μπορούμε επίσης να θέσουμε κάποια bits σε έναν καταχωρητή με την μακροεντολή sbi(num,bitloc), να μηδενίσουμε με την cbi(num,bitloc). Οι ορισμοί P00 έως P31 χρησιμοποιούνται για τον χειρισμό των πινς των θυρών εισόδου και εξόδου.

7.5 Το αρχείο “armlibtypes.h”

Το αρχείο armlibtypes.h είναι ένα αρχείο επικεφαλίδας που ορίζει πιο εύκολα ονόματα για τους τύπους μεταβλητών που χρησιμοποιούνται. Αυτό είναι:

```
#ifndef ARMLIBTYPES_H
#define ARMLIBTYPES_H

#define FALSE       0
#define TRUE        -1
#define OK          0
#define ERROR       1

// datatype definitions macros
typedef unsigned char           u08;
typedef   signed char           s08;
typedef unsigned short          u16;
typedef   signed short          s16;
typedef unsigned long           u32;
typedef   signed long           s32;
typedef unsigned long long      u64;
typedef   signed long long      s64;

// maximum value that can be held
```

```
// by unsigned data types (8,16,32bits)
#define MAX_U08   255
#define MAX_U16   65535
#define MAX_U32   4294967295

// maximum values that can be held
// by signed data types (8,16,32bits)
#define MIN_S08    -128
#define MAX_S08    127
#define MIN_S16    -32768
#define MAX_S16    32767
#define MIN_S32    -2147483648
#define MAX_S32    2147483647

// more type redefinitions
typedef unsigned char           BOOL;
typedef unsigned char           BYTE;
typedef unsigned short          WORD;
typedef unsigned long           DWORD;

typedef unsigned char           UCHAR;
typedef unsigned int            UINT;
typedef unsigned short          USHORT;
typedef unsigned long           ULONG;

typedef char                    CHAR;
typedef int                     INT;
typedef long                    LONG;

// typedefs are here
#include <stdint.h>

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;

#endif
```

Αρχικά ορίζονται τα FALSE, TRUE, OK και ERROR ώστε να κάνουν το πρόγραμμα πιο ευανάγνωστο. Στην συνέχεια ορίζονται νέα ονόματα για τους τύπους μεταβλητών που διευκολύνουν τον ορισμό μεταβλητών ώστε να καθορίζουμε ακριβώς τις μεταβλητές με το μέγεθος που χρειαζόμαστε. Για παράδειγμα αν θέλουμε μια μεταβλητή να έχει μέγεθος 8bits και να παίρνει θετικές τιμές από 0 έως 255, την ορίζουμε ώς u08. Το u σημαίνει unsigned, δηλαδή χωρίς πρόσημο και το 08 είναι ο αριθμός των bits που έχει. Αν

θέλουμε να παίρνει και αρνητικές τιμές χρειάζεται πρόσημο δηλαδή γίνεται s08 (signed). Με αυτό τον τρόπο δεν χρειάζεται να θυμόμαστε πόσα bits έχει για παράδειγμα η unsigned long μεταβλητή και συνεπώς διευκολύνεται η συγγραφή του κώδικα.

7.6 Το αρχείο “board.h”

Το αρχείο board.h περιέχει ορισμούς για το υλικό (hardware) που επικοινωνεί με τον μικροελεγκτή της πλακέτας.

```
#ifndef  BOARD_H
#define  BOARD_H

#define BUTTON_1          P30
#define LED_1             P21
#define LED_2             P22

#define BUTTON_1_PRESSED   ( !(IOPIN0 & BUTTON_1) )
#define LED_1_ON           ( IOCLR0 = LED_1 )
#define LED_2_ON           ( IOCLR0 = LED_2 )
#define LEDS_ON            ( IOCLR0 = LED_1 | LED_2 )
#define LED_1_OFF          ( IOSET0 = LED_1 )
#define LED_2_OFF          ( IOSET0 = LED_2 )
#define LEDS_OFF           ( IOSET0 = LED_1 | LED_2 )

#endif
```

Σε αυτό ορίζονται σε ποια πινς εισόδου/εξόδου είναι συνδεδεμένα το κουμπί1 και τα leds 1 και 2. Στην συνέχεια ορίζονται μακροεντολές για την ανάνωση του κουμπιού και για το άναμμα και σβύσιμο των leds.

7.7 Το αρχείο “delays.c”

Στο delays.c ορίζεται μια απλή συνάρτηση καθυστέρησης.

```
#include "includes.h"

void delay(u32 d)
{
        for(; d; --d)
        {
                asm volatile ("nop");
        }
}
```

Η delay() παίρνει για όρισμα τον αριθμό των εντολών “nop” που θα εκτελεσθούν. Η συνάρτηση χρησιμοποιεί το πρόθεμα volatile στη “nop” επειδή ο μεταγλωττιστής αλλιώς μπορεί να δει ότι δεν υπάρχει κώδικας που να φέρει κάποια αλλαγή στο πρόγραμμα και να τον παραλείψει.

7.8 Τα αρχεία “uart.h” και “uart.c”

Η σειριακή επικοινωνία βασίζεται στα αρχεία “uart.h” και “uart.c”. Τα αρχεία αυτά ορίζουν συναρτήσεις τόσο για το UART0 όσο και για το UART1 του μικροελεγκτή. Ο κώδικας για το UART0 είναι ο εξής:

```
#define CR     0x0D

volatile u08 UART0Buffer[BUFSIZE_UART0];
volatile u16 wr_index_uart0,rd_index_uart0,counter_uart0;
volatile u08 buffer_overflow_uart0;

// Initialize Serial Interface UART0
void init_UART0 ( unsigned long baudrate )
{
        unsigned long Fdiv;

        Fdiv = (60000000/16) / baudrate;

        PINSEL0      =  ((PINSEL0 & 0xFFFFFFF0) | 0x00000005);
        U0IER= 0;

        U0LCR        = (1<<7);
        U0DLM        = Fdiv / 256;
        U0DLL        = Fdiv % 256;
        U0LCR        = 0x03;
        U0FCR        = 0x07;

        // Xwris interrupt to UART0
        /*
        VICIntSelect &= ~BIT(6);
        VICVectCntl1 = 0x20 | 6;
        VICVectAddr1 = (DWORD)UART_0_Handler;
        VICIntEnable = BIT(6);

        rd_index_uart0 = 0;
        wr_index_uart0 = 0;
        counter_uart0 = 0;
        U0IER = BIT(0);    //receive RBR interrupt
        */
}

// UART_0_Handler INTERRUPT RBR WITH BUFFER
void UART_0_Handler (void)
{

        // Receive Data Available
        UART0Buffer[wr_index_uart0] = U0RBR;
        wr_index_uart0++;
        if(wr_index_uart0 == BUFSIZE_UART0)
```

```
                wr_index_uart0 = 0;
        counter_uart0++;
        if(counter_uart0 == BUFSIZE_UART0)
        {
                counter_uart0 = 0;
                buffer_overflow_uart0 = 1;
        };

   VICVectAddr = 0;          // Acknowledge Interrupt
}

// Write character to Serial Port 0 with \n -> \r\n
char sendchar_UART0 (char ch)
{
        if (ch == '\n')  {
                while (!(U0LSR & 0x20));            //wait until U0THR is empty
                U0THR = CR;                 // output CR Carriage Return
        }
        while (!(U0LSR & 0x20));
        return (U0THR = ch);
}

// Write character to Serial Port 0 without \n -> \r\n
int sendc_UART0 (u08 ch)
{
        while (!(U0LSR & 0x20));
        return (U0THR = ch);
}

void sendstring_UART0 (const char *string)
{
        char ch;

        while ((ch = *string)) {
                sendchar_UART0(ch);
                string++;
        }
}

// Read character from Serial Port
char getc_UART0 (void)
{
        if (U0LSR & 0x01) {
                return (U0RBR);
        }
        else {
                return 0;
        }
}
```

```
int getstring_UART0 (char *buf, int length)
{
        int i;
        char sertemp;

        for(i=0;i<length;i++)
        {
                while( (U0LSR & 0x01)==0) ;
                sertemp = U0RBR;
                if(sertemp == 0x0D)          //0x0D == CR
                {
                        *buf='\0';
                        return i;
                }
                *buf=sertemp;
                buf++;
        }
        return i;
}

// send data - NOT only string
void send_data_UART0(u08 *BufferPtr, u32 Length )
{
   while ( Length != 0 )
   {
        while (!(U0LSR & 0x20));   //wait until ...
        U0THR = *BufferPtr;
        BufferPtr++;
        Length--;
   }
   return;
}
/*****************************************************************************
** Function name:              UART0_getchar()
   Reads data from UART0Buffer[]
*****************************************************************************/
u08 UART0_getchar(void)
{
        u08 data2;

        //while(rx_counter==0) ;
        data2 = UART0Buffer[rd_index_uart0];
        if(++rd_index_uart0 == BUFSIZE_UART0)
                rd_index_uart0=0;
        VICIntEnClr = BIT(6); //disable UART0 interrupt
        counter_uart0--;
        VICIntEnable = BIT(6); //enable UART0 interrupt

        return data2;
}
```

Αρχικά ορίζεται η συνάρτηση αρχικοποίησης του περιφερειακού UART0. Δέχεται για όρισμα της την ταχύτητα μεταφοράς δεδομένων και προγραμματίζει τους καταχωρητές του περιφερεικού για να έχουμε αυτή την ταχύτητα. Επίσης ορίζει τα πινς του μικροελεγκτή σε λειτουργία uart και ρυθμίζει τον καταχωρητή U0LCR ώστε ο τρόπος μεταφοράς να είναι 8bits, ένα stop bit και χωρίς ισοτιμία.

Επίσης υπάρχει και ο κώδικας αν θέλουμε να έχουμε λειτουργία UART με διακοπές. Ορίζουμε έναν κυκλικό χώρο αποθήκευσης UART0Buffer[] που όταν έρχεται κάποιο δεδομένο γίνεται διακοπή και γράφεται σε αυτόν. Όταν θέλουμε να διαβάσουμε από τον buffer χρησιμοποιούμε την συνάρτηση UART0_getchar(). Υπάρχουν και οι ανάλογοι δείκτες που δείχνουν σε ποια θέση του buffer θα γράφτει το επόμενο δεδομένο, ως ποια θέση έχουμε διαβάσει και πόσα δεδομένα ακόμα μένουν να διαβαστούν. Με αυτό τον τρόπο μπορούμε να λαμβάνουμε δεδομένα χωρίς να διακόπτουμε το πρόγραμμά μας για να καλέσουμε μια συνάρτηση ανάγνωσης. Ωστόσο επειδή εμείς κυρίως στέλνουμε δεδομένα μέσω της σειριακής θύρας δεν χρησιμοποιήσαμε αυτή την μέθοδο.

Επιπλέον ορίζονται συναρτήσεις όπως η sendc_UART0() η οποία στέλνει ένα byte μέσω της σειριακής θύρας, η sendstring_UART0() στέλνει μια συμβολοσειρά και η send_data_UART0() στέλνει έναν συγκεκριμένο αριθμό bytes από έναν πίνακα που περνάνε σε αυτήν μέσω ορισμάτων. Υπάρχουν επίσης οι κατάλληλες συναρτήσεις λήψης χαρακτήρων με την μέθοδο της κυκλικής ανίχνευσης (polling). Με τον ιδιο τρόπο ορίζονται συναρτήσεις για το UART1. Για μορφοποιημένη έξοδο δεδομένων υπάρχει η συνάρτηση rprintf() η οποία ορίζεται στα αρχεία “rprintf.h” και “rprintf.c”.

7.9 Τα αρχεία “i2c.c” και “i2c.h”

Για το περιφερειακό I2C του μικροελεγκτή χρησιμοποιούνται τα αρχεία “i2c.h” και “i2c.c”. Το αρχείο επικεφαλίδας “i2c.h” εκτός από τα πρότυπα των συναρτήσεων, καθορίζει και τους κώδικες κατάστασης που εμφανίζονται στον διάδρομο i2c και μπορούν να διαβαστούν από τον καταχωρητή κατάστασης i2c. Ένα μέρος του αρχείου αυτού είναι:

```
/* I2C values */
/* Master */
#define TW_START                0x08
#define TW_REP_START            0x10
/* Master Transmitter */
#define TW_MT_SLA_ACK           0x18
#define TW_MT_SLA_NACK          0x20
#define TW_MT_DATA_ACK          0x28
#define TW_MT_DATA_NACK         0x30
#define TW_MT_ARB_LOST          0x38
/* Master Receiver */
#define TW_MR_ARB_LOST          0x38
#define TW_MR_SLA_ACK           0x40
```

```
#define TW_MR_SLA_NACK                  0x48
#define TW_MR_DATA_ACK                  0x50
#define TW_MR_DATA_NACK                 0x58
/* Slave Transmitter */
#define TW_ST_SLA_ACK                   0xA8
#define TW_ST_ARB_LOST_SLA_ACK          0xB0
#define TW_ST_DATA_ACK                  0xB8
#define TW_ST_DATA_NACK                 0xC0
#define TW_ST_LAST_DATA                 0xC8
/* Slave Receiver */
#define TW_SR_SLA_ACK                   0x60
#define TW_SR_ARB_LOST_SLA_ACK          0x68
#define TW_SR_GCALL_ACK                 0x70
#define TW_SR_ARB_LOST_GCALL_ACK        0x78
#define TW_SR_DATA_ACK                  0x80
#define TW_SR_DATA_NACK                 0x88
#define TW_SR_GCALL_DATA_ACK            0x90
#define TW_SR_GCALL_DATA_NACK           0x98
#define TW_SR_STOP                      0xA0
/* Misc */
#define TW_NO_INFO                      0xF8
#define TW_BUS_ERROR                    0x00

// bit defines
#define I2CON_AA                2
#define I2CON_SI                3
#define I2CON_STO               4
#define I2CON_STA               5
#define I2CON_I2EN              6

// defines and constants
#define TWCR_CMD_MASK           0x0F
#define TWSR_STATUS_MASK        0xF8

// return values
#define I2C_OK                  0x00
#define I2C_ERROR_NODEV         0x01
```

Στο “i2c.c” ορίζονται οι συναρτήσεις που χρησιμοποιούνται για την επικοινωνία με το i2c περιφερειακό. Αυτές είναι:

```
void i2cInit(void)
{
        // setup SCL pin P02
        PINSEL0 &= ~(3<<4);
        PINSEL0 |=   1<<4;

        // setup SDA pin P03
        PINSEL0 &= ~(3<<6);
```

```
        PINSEL0 |=   1<<6;
        // set bitrate
        i2cSetBitrate(75);  // 60MHz - 400kHz

        // disable and reset interface
        I20CONCLR = 0xFF;
        delay(10);
        // enable interface
        I20CONSET = BIT(I2CON_I2EN);
}
```

Η i2cInit() αρχικοποιεί το περιφερειακό. Αρχικά θέτει τα πινς του μικροελεγκτή σε λειτουργία i2c δηλαδή ενεργοποιεί τις γραμμές SDA και SCL που χρησιμοποιεί ο διάδρομος i2c. Στην συνέχεια καλεί την συνάρτηση i2cSetBitrate() και καθορίζει την συχνότητα επικοινωνίας στα 400kHz και τέλος ενεργοποιεί το περιφερειακό i2c μέσω του καταχωρητή ελέγχου.

```
void i2cSetBitrate(u16 bitrateDiv)
{
        // @60MHz and VPB=1, set to 75 for 400KHz
        // set equal high and low periods
        I20SCLL = bitrateDiv;
        I20SCLH = bitrateDiv;
}
```

Η συνάρτηση i2cSetBitrate() καθορίζει την συχνότητα επικοινωνίας στον διάδρομο i2c γράφοντας τις κατάλληλες τιμές στους καταχωρητές I20SCLL και I20SCLH. Η συχνότητα επικοινωνίας προκύπτει Fi2c=Pclk/(I20SCLL+ I20SCLH) δηλαδή Fi2c=60000000/150=400kHz.

```
void i2cSendStart(void)
{
        I20CONSET = BIT(I2CON_STA);
        I20CONCLR = BIT(I2CON_SI);
}

void i2cSendStop(void)
{
        I20CONSET = BIT(I2CON_STO);
        I20CONCLR = BIT(I2CON_SI);
}
```

Η συνάρτηση i2cSendStart() δημιουργεί μια START κατάσταση στον διάδρομο i2c ενώ η i2cSendStop() δημιουργεί μια STOP κατάσταση. Και οι δύο συναρτήσεις ενεργοποιούν συγκεκριμένα bits του καταχωρητή ελέγχου i2c που σχετίζονται με την επιθυμητή λειτουργία.

```
void i2cWaitForComplete(void)
{
	// wait for a valid status code
	while(I20STAT == TW_NO_INFO);
}
```

Η συνάρτηση i2cWaitForComplete() διαβάζει συνεχώς τον καταχωρητή κατάστασης i2c και περιμένει να δημιουργηθεί μια κατάσταση i2c από αυτές που έχουν οριστεί στο αρχείο “i2c.h”. Χρησιμοποιείται για να μάθουμε σε τι κατάσταση βρίσκεται ο διάδρομος, αν τα δεδομένα που στείλαμε λήφθηκαν με επιτυχία και γενικά για να καθορίσουμε την επόμενη λειτουργία του περιφερειακού.

```
void i2cSendByte(u08 data)
{
	// save data into data register
	I20DAT = data;
	// clear SI bit to begin transfer
	I20CONCLR = BIT(I2CON_SI);
}
```

Η i2cSendByte() στέλνει το byte που δέχεται στο όρισμά της γράφοντάς το στον καταχωρητή δεδομένων i2c.

```
u08 i2cGetReceivedByte(void)
{
	return I20DAT;
}
```

Η i2cGetReceivedByte() διαβάζει το byte που έχει ληφθεί από το περιφερειακό.

```
void i2cReceiveByte(u08 ackFlag)
{
	// begin receive over i2c
	if( ackFlag )
	{
		// ackFlag = TRUE: ACK the recevied data
		I20CONSET = BIT(I2CON_AA);
	}
	else
	{
		// ackFlag = FALSE: NACK the recevied data
		I20CONCLR = BIT(I2CON_AA);
	}
	// clear SI bit to begin transfer
	I20CONCLR = BIT(I2CON_SI);
}
```

Η i2cReceiveByte() δημιουργεί ACK ή NACK κατάσταση ανάλογα με το όρισμά της ώστε να συνεχίσει ή να σταματήσει την επικοινωνία με την άλλη συσκευή i2c αφού λάβει το byte που μας έστειλε.

```
u08 i2cGetStatus(void)
{
	return I20STAT;
}
```

Η i2cGetStatus() επιστρέφει την κατάσταση i2c που δίνεται από τον καταχωρητή κατάστασης.
Οι δύο ανωτέρου επιπέδου συναρτήσεις μεταφοράς είναι οι εξής:

```
u08 i2cMasterSendNI(u08 deviceAddr, u08 length, u08* data)
{
	u08 retval = I2C_OK;

	// send start condition
	i2cSendStart();
	i2cWaitForComplete();
	I20CONCLR = BIT(I2CON_STA);

	// send device address with write
	i2cSendByte( deviceAddr & 0xFE );
	i2cWaitForComplete();

	// check if device is present and live
	if( I20STAT == TW_MT_SLA_ACK)
	{
		// send data
		while(length)
		{
			i2cSendByte( *data++ );
			i2cWaitForComplete();
			length--;
		}
	}
	else
	{
		// device did not ACK it's address,
		// data will not be transferred
		// return error
		retval = I2C_ERROR_NODEV;
	}
	// transmit stop condition
	i2cSendStop();

	return retval;
}
```

```
u08 i2cMasterReceiveNI(u08 deviceAddr, u08 length, u08 *data)
{
	u08 retval = I2C_OK;

	// send start condition
	i2cSendStart();
	i2cWaitForComplete();
	I20CONCLR = BIT(I2CON_STA);

	// send device address with read
	i2cSendByte( deviceAddr | 0x01 );
	i2cWaitForComplete();

	// check if device is present and live
	if( I20STAT == TW_MR_SLA_ACK)
	{
		// accept receive data and ack it
		while(length > 1)
		{
			i2cReceiveByte(TRUE);
			i2cWaitForComplete();
			*data++ = i2cGetReceivedByte();
			// decrement length
			length--;
		}
		// accept receive data and nack it (last-byte signal)
		i2cReceiveByte(FALSE);
		i2cWaitForComplete();
		*data++ = i2cGetReceivedByte();
	}
	else
	{
		// device did not ACK it's address,
		// data will not be transferred
		// return error
		retval = I2C_ERROR_NODEV;
	}
	// transmit stop condition
	// leave with TWEA on for slave receiving
	i2cSendStop();
	return retval;
}
```

Η συνάρτηση i2cMasterSendNI() στέλνει στην συσκευή με διεύθυνση deviceAddr, length αριθμό bytes τα οποία αρχίζουν από την διεύθυνση data που μπορεί να είναι ένας πίνακας. Αν η συσκευή δεν αναγνωρίσει το byte διεύθυνσης που στέλνουμε στην αρχή, δηλαδή δεν δημιουργήσει ACK κατάσταση, τότε η συνάρτηση επιστρέφει λάθος, I2C_ERROR_NODEV, αλλιώς επιστρέφει I2C_OK.

Ομοίως η συνάρτηση i2cMasterReceiveNI() διαβάζει length αριθμό bytes από την συσκευή deviceAddr και τα αποθηκεύει στον πίνακα data. Κάθε byte που λαμβάνει το αναγνωρίζει στέλνοντας ACK εκτός του τελευταίου που στέλνει NACK για να σταματήσει την επικοινωνία με την άλλη συσκευή i2c.

7.10 Τα αρχεία “max1237.h” και “max1237.c”

Για την επικοινωνία με το ολοκληρωμένο max1237 χρησιμοποιούμε τα αρχεία “max1237.h” και “max1237.c”. Το “max1237.h” είναι το αρχείο επικεφαλίδας :

```
#ifndef MAX1237_H_
#define MAX1237_H_

#define MAX1237_ADDRESS          0x68   //lsb 0(R/W) 0b0110100x
#define SETUP_BYTE               0x82
#define CONFIG_BYTE              0x05
extern void max1237_init(void);
extern u08  max1237_read(u08 *adcdata);
#endif
```

Σε αυτό αρχικά ορίζεται η διεύθυνση i2c του ολοκληρωμένου MAX1237_ADDRESS που χρειάζεται για να αρχίσουμε την επικοινωνία μαζί του. Επίσης ορίζεται το Setup byte ώστε το max1237 να χρησιμοποιεί για τάση αναφοράς στις μετατροπές που κάνει την τάση τροφοδοσίας VDD. Ακόμα ρυθμίζεται ώστε να χρησιμοποιεί εσωτερικό ρολόι, μονοπολικό τρόπο λειτουργίας και επανεκκινεί τον Configuration καταχωρητή.

Το Config byte καθορίζεται ώστε το max1237 να διαβάζει τις αναλογικές εισόδους AIN0 έως AIN2 διαδοχικά και καθορίζει τα κανάλια να είναι μονά και όχι διαφορικά. Στην συνέχεια ορίζονται τα πρωτότυπα των συναρτήσεων που υπάρχουν στο αρχείο “max1237.c”.

```
#include "includes.h"
//########################  max1237_init  ############################
void max1237_init (void)
{
u08 initdata[2]={SETUP_BYTE , CONFIG_BYTE};

        i2cMasterSendNI(MAX1237_ADDRESS, 2, initdata);
}
//#####################  max1237_read  ############################
u08  max1237_read (u08 *adcdata)
{
u08 retvaladc;

        retvaladc = i2cMasterReceiveNI(MAX1237_ADDRESS, 6, adcdata);

return retvaladc;
}
```

Επειδή αυτές οι συναρτήσεις χρησιμοποιούν τις συναρτήσεις που έχουν οριστεί στο αρχείο “i2c.c” πρέπει να αρχικοποιηθεί πρώτα ο i2c διάδρομος με την συνάρτηση i2cInit(). Η max1237_init() στέλνει στο ολοκληρωμένο με την διεύθυνση MAX1237_ADDRESS τα Setup και Configuration bytes που έχουν οριστεί στο “max1237.h”. Μετά από το κάλεσμα αυτής της συνάρτησης το max1237 έχει ρυθμιστεί να έχει την λειτουργία που θέλουμε.

Όταν θέλουμε το max1237 να κάνει μετατροπή των αναλογικών σημάτων και να διαβάσουμε τα δεδομένα που παράγει, καλούμε την συνάρτηση max1237_read() με όρισμα έναν δείκτη σε μια θέση μνήμης που έχουμε δεσμεύσει που να χωράει τα αποτελέσματα (πχ. έναν πίνακα) και διαβάζουμε έξι bytes (δύο για κάθε αναλογική είσοδο αφού το max1237 έχει 10bits ακρίβεια).

7.11 Το αρχείο “includes.h”

Στο αρχείο “includes.h” περιλαμβάνονται όλα τα αρχεία επικεφαλίδας που χρησιμοποιούμε ώστε να τα έχουμε συγκεντρωμένα και να μην γίνονται λάθη κατά την μεταγλώττιση του κώδικα.

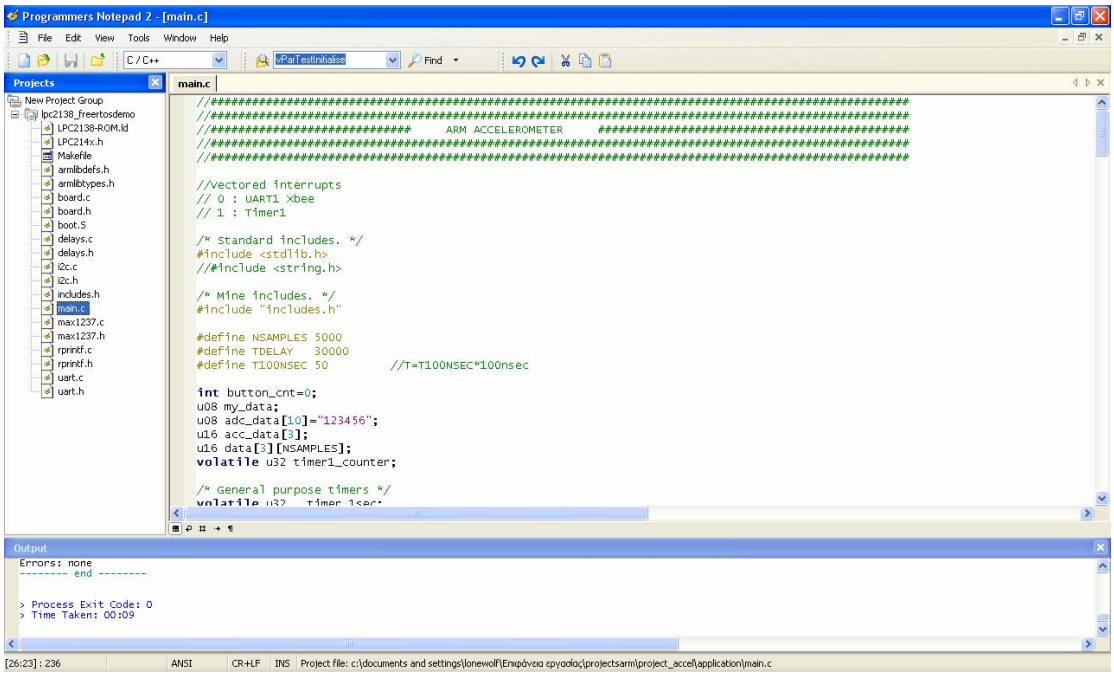
```
#ifndef includes_h_
#define includes_h_

#include "LPC214x.h"
#include "armlibtypes.h"
#include "armlibdefs.h"
#include "board.h"
#include "uart.h"
#include "i2c.h"
#include "rprintf.h"
#include "delays.h"
#include "max1237.h"
#endif
```

ΚΕΦΑΛΑΙΟ 8

Η διαδικασία προγραμματισμού του μικροελεγκτή

Για τον προγραμματισμό του μικροελεγκτή LPC2148 χρησιμοποιήσαμε τον μεταγλωττιστή GCC (GNU Compiler Collection) για τους ARM μικροελεγκτές. Το περιβάλλον στο οποίο γράψαμε κώδικα ήταν το WinARM και συγκεκριμένα σαν συντάκτης κειμένου χρησιμοποιήθηκε το Programmers Notepad.

Αρχικά γράφουμε τον κώδικα εφαρμογής σε γλώσσα C και σε γλώσσα μηχανής στο Programmer's Notepad.

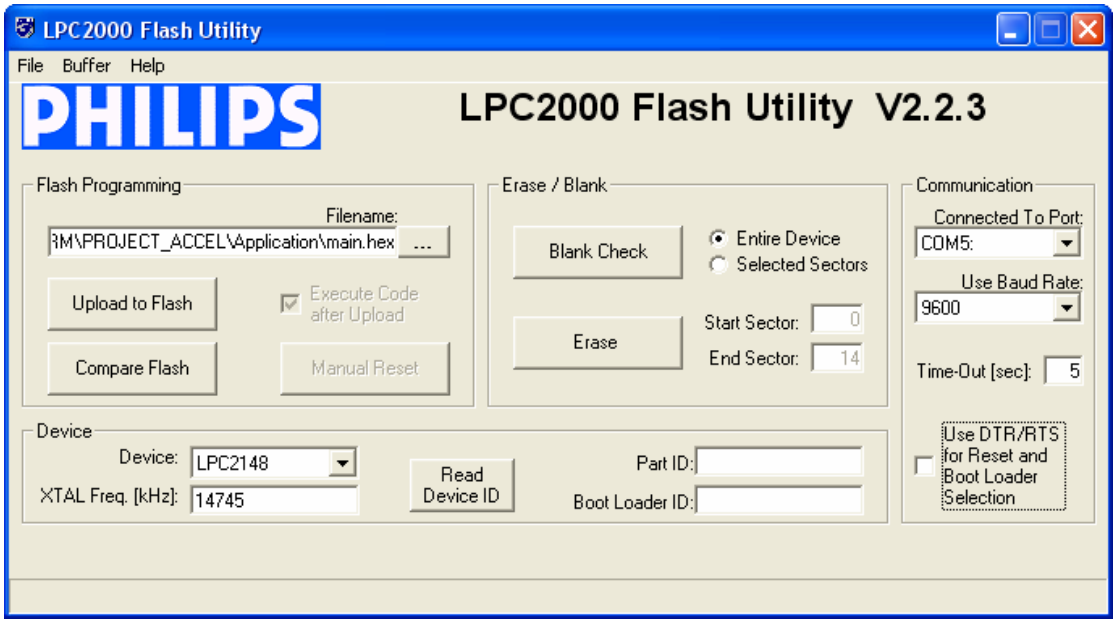


Σχήμα 8.1: Το πρόγραμμα Programmer's Notepad.

Στην συνέχεια κάνουμε μεταγλώττιση με την εντολή "Tools->Make All". Αν δεν υπάρχουν λάθη στον κώδικά μας, στο παράθυρο Output θα εμφανιστεί ένα κείμενο που παρουσιάζει τα αρχεία που δημιουργήθηκαν και το μέγεθος του κώδικα χωρισμένος σε τμήματα και τις διευθύνσεις που θα έχουν στον μικροελεγκτή (κώδικα αρχικοποίησης, κώδικα εφαρμογής, μεταβλητές).

Σχήμα 8-2: Η έξοδος του προγράμματος μετά τη μεταγλώττισή του προγράμματος. Μας δείχνει την μνήμη που καταναλώνει το πρόγραμμα και αν υπήρχαν λάθη στην μεταγλώττιση.

Στην συνέχεια για να περάσουμε τον κώδικα στον μικροελεγκτή ανοίγουμε το πρόγραμμα LPC2000 Flash Utility.



Σχήμα 8-3: Το πρόγραμμα LPC2000 Flash Utility μεταφέρει τον κώδικα στον μικροελεγκτή.

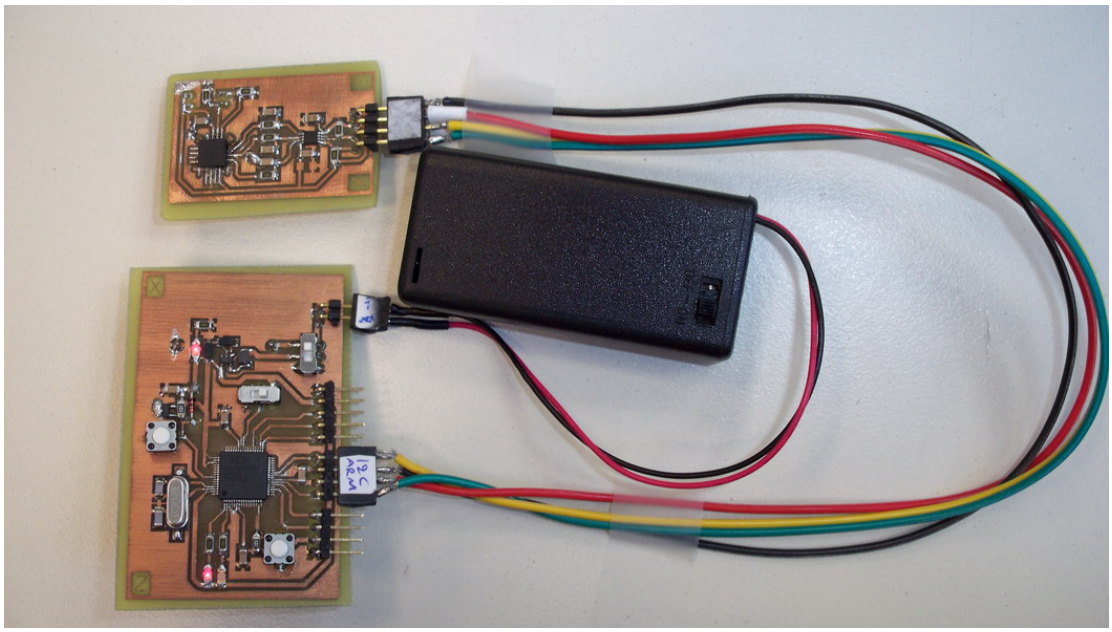
Στο πεδίο Filename επιλέγουμε το αρχείο που θα περάσουμε στον μικροελεγκτή. Αυτό στην περίπτωσή μας είναι το “main.hex”. Στο πεδίο Device επιλέγουμε τον τύπο του μικροελεγκτή που χρησιμοποιούμε (LPC2148) και στο πεδίο Communication επιλέγουμε την σειριακή θύρα του υπολογιστή με την οποία θα επικοινωνήσουμε με τον μικροελεγκτή και την ταχύτητα επικοινωνίας. Εμείς χρησιμοποιούμε την θύρα 5 σε ταχύτητα 9600bps. Οταν συνδέσουμε τον μικροελεγκτή στον υπολογιστή και ενεργοποιήσουμε τον εσωτερικό του bootloader, πατάμε το κουμπί Upload to Flash και το πρόγραμμα αρχίζει να μεταφέρεται στον LPC2148.

ΚΕΦΑΛΑΙΟ 9

Διαδικασία συλλογής και επεξεργασίας δεδομένων

9.1 Συλλογή δειγμάτων

Η συλλογή δεδομένων έγινε με τις πλακέτες που κατασκευάστηκαν. Αρχικά τοποθετούμε την πλακέτα με τον αισθητήρα επιτάχυνσης και τον αναλογικοψηφιακό μετατροπέα (ADC) στο σημείο του ποδιού που θέλουμε να μετρήσουμε τις επιταχύνσεις. Στην συνέχεια συνδέουμε τα καλώδια τροφοδοσίας και της διασύνδεση I2C στην πλακέτα με τον μικροελεγκτή. Την πλακέτα με τον μικροελεγκτή την τοποθετούμε σε ένα σταθερό σημείο στο πόδι. Όταν είμαστε έτοιμοι ανοίγουμε την τροφοδοσία και περιμένουμε να αρχικοποιηθεί το κύκλωμα. Στη συνέχεια πατάμε το κουμπί της πλακέτας και περιμένουμε να ανάψει το led “ένα” που σημαίνει ότι η δειγματοληψία έχει ξεκινήσει και εμείς μπορούμε να αρχίσουμε να περπατάμε. Η συνόλική διάρκεια των μετρήσεων εξαρτάται από την συχνότητά της δειγματοληψίας και τα δείγματα που παίρνουμε. Αυτά έχουν καθορίστεί στον κώδικα που γράψαμε στον μικροελεγκτή. Συγκεκριμένα η συχνότητα δειγματοληψίας είναι περίπου 200Hz (δηλαδή T=5msec) και τα δείγματα είναι 5000 ανά άξονα. Αυτό σημαίνει ότι η συνολική διάρκεια των μετρήσεων είναι 5000*5msec=25sec. Μόλις τελειώσει ο χρόνος μετρήσεων ανάβει και το δεύτερο led.



Σχήμα 9.1: Το σύστημα ολοκληρωμένο, έτοιμο να το τοποθετήσουμε στο πόδι.

Σχήμα 9-2: Το σύστημα τοποθετημένο στο πόδι.

Σχήμα 9-3: Το υποσύστημα επιταχυνσιομέτρου τοποθετημένο στο πόδι.

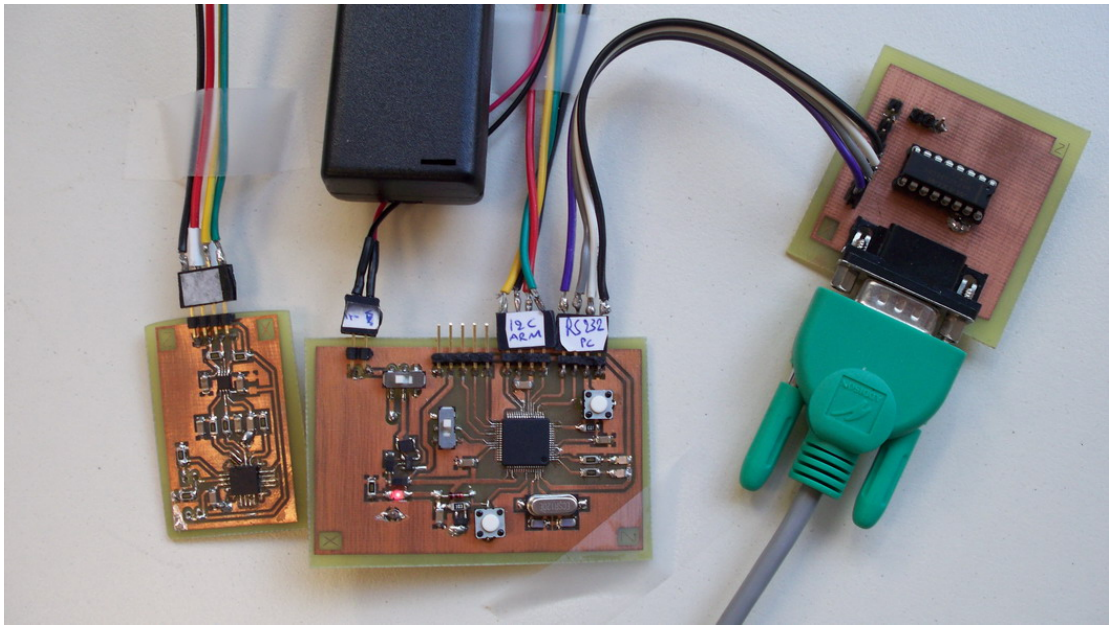
Σχήμα 9-4: Το σύστημα τοποθετημένο στο πόδι.

Σχήμα 9-5: Το υποσύστημα επιταχυνσιομέτρου τοποθετημένο στο πόδι.

Σχήμα 9-6: Το σύστημα τοποθετημένο στο πόδι.

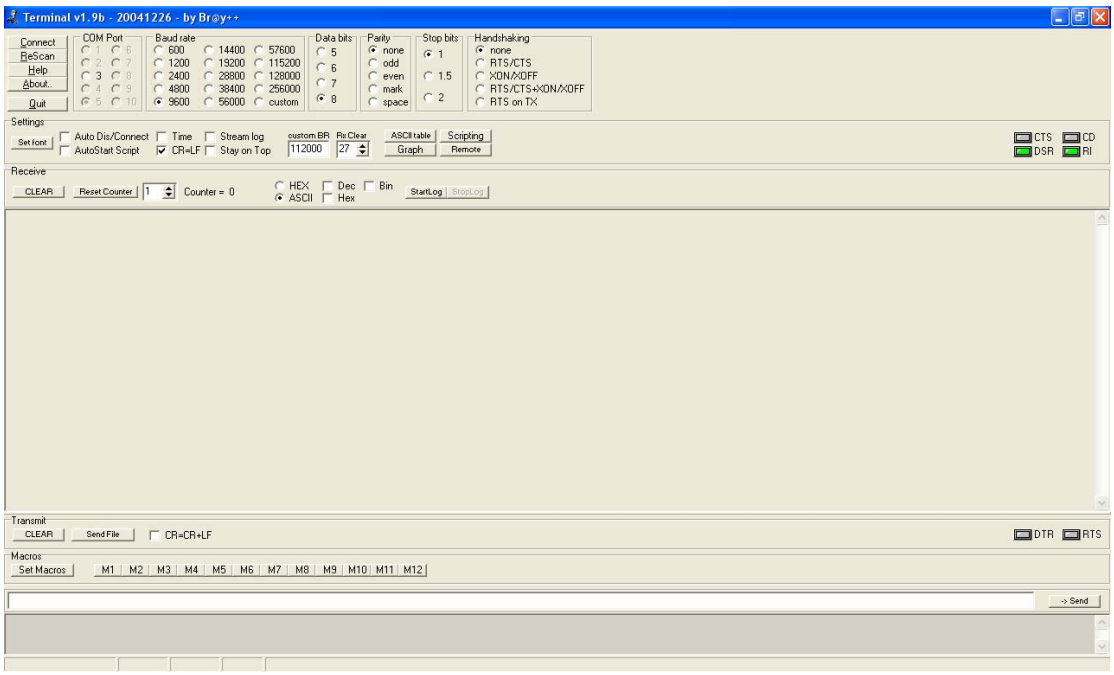
9.2 Μεταφορά δειγμάτων στον υπολογιστή

Για να περάσουμε τα δείγματα που λάβαμε στον προσωπικό υπολογιστή χρησιμοποιούμε την σειριακή θύρα. Επειδή η σειριακή θύρα του υπολογιστή λειτουργεί σε τάσεις 12V ενώ ο μικροελεγκτής στα 3.3V συνδέουμε τον μικροελεγκτή σε μια άλλη πλακέτα που κατασκευάσαμε η οποία μετατρέπει τα σήματα που παίρνει από τον μικροελεγκτή από τα 3.3V στα 12V και τα τροφοδοτεί στον υπολογιστή. Ανάλογη λειτουργία έχει για την αντίθετη φορά δεδομένων. Το ολοκληρωμένο που χρησιμοποιεί αυτή η πλακέτα είναι το MAX3232 και χρειάζεται μόνο 3.3V στην τάση τροφοδοσίας που την παίρνει από την πλακέτα του μικροελεγκτή.

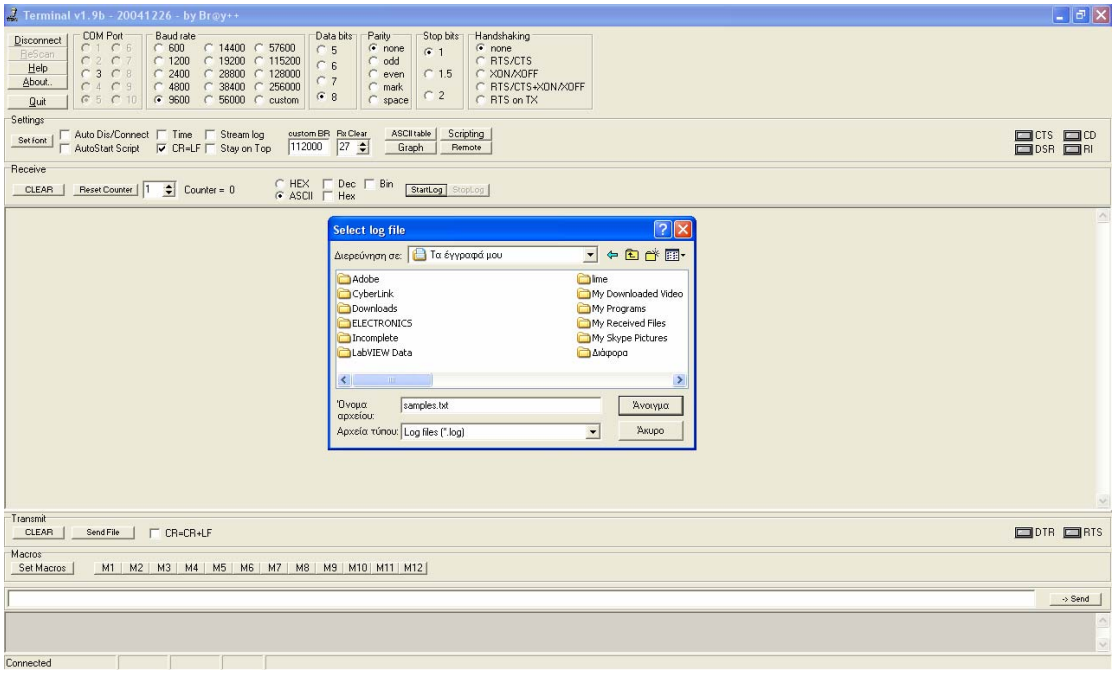


Σχήμα 9-7: Το σύστημα μαζί με τον μετατροπέα τάσης MAX3232 με συνδεδεμένο το καλώδιο RS232 και έτοιμο για σύνδεση με τον υπολογιστή.

Το πρόγραμμα που αναλαμβάνει να λάβει και να αποθηκεύσει τα σειριακά δεδομένα στον υπολογιστή είναι το “Bray’s Terminal”. Σε αυτό ορίζουμε την σειριακή θύρα που χρησιμοποιούμε στον υπολογιστή, την ταχύτητα μεταφοράς δεδομένων στα 9600bps, 8 μπιτ δεδομένων, καμία ισοτιμία και ένα στοπ μπιτ. Στην συνέχεια πατάμε το κουμπί “connect” και μετά το “Start log” στο πρόγραμμα και καθορίζουμε το όνομα του αρχείου που θα αποθηκευτούν τα δεδομένα.

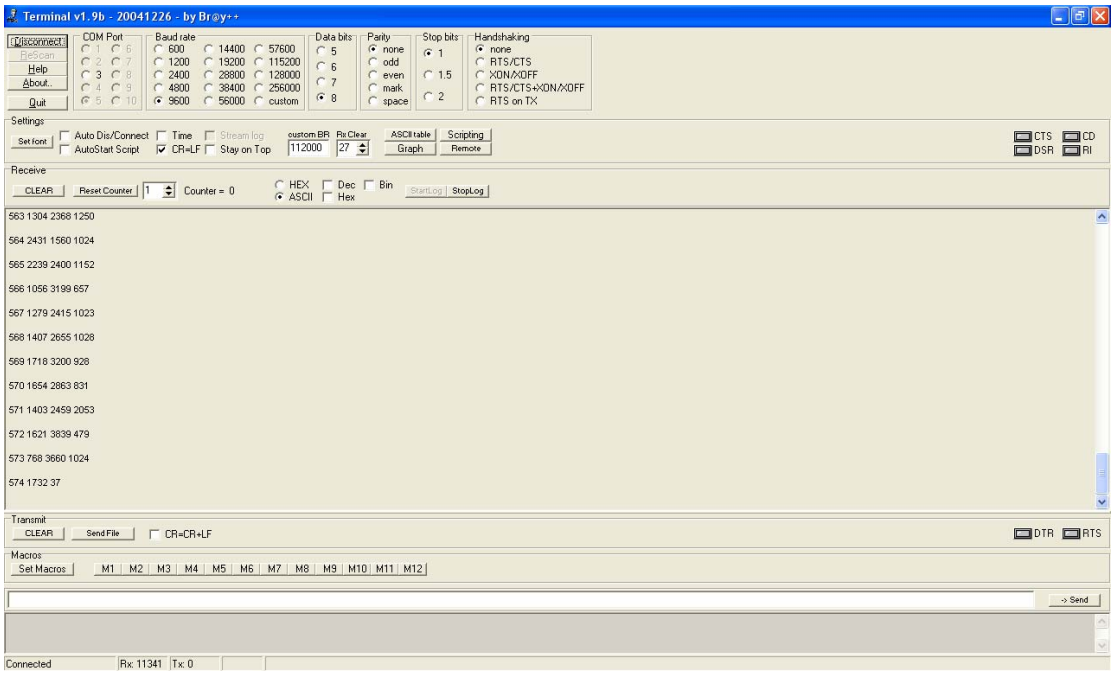


Σχήμα 9-8: Το πρόγραμμα Bray’s Terminal.

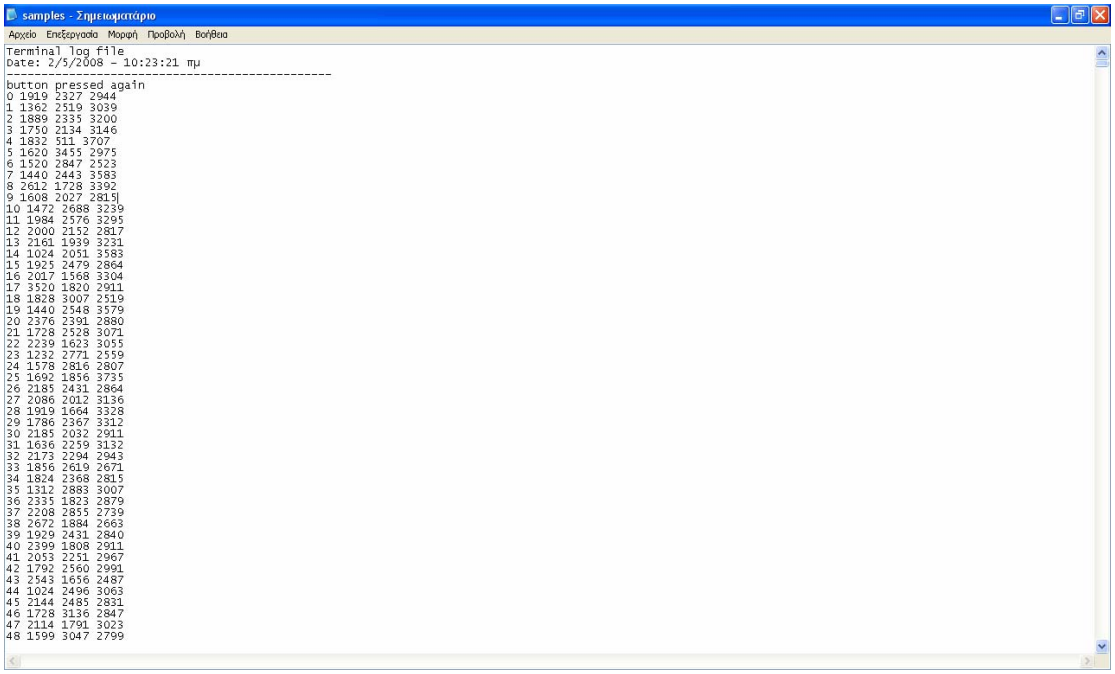


Σχήμα 9-9: Καθορισμός του αρχείου αποθήκευσης δεδομένων στο πρόγραμμα Bray's Terminal.

Μόλις είμαστε έτοιμοι πατάμε το κουμπί "ένα" στην πλακέτα του μικροελεγκτή και αυτός στέλνει σειριακά τα δεδομένα στον υπολογιστή. Μόλις τελειώσει η αποστολή πατάμε το "Stop log" στο πρόγραμμα. Τα δεδομένα έχουν αποθηκευτεί σε τέσσερις στήλες. Στην πρώτη στήλη είναι ο αριθμός του δείγματος, στην δεύτερη η ψηφιακή τιμή του σήματος επιτάχυνσης του "x" άξονα, στην τρίτη του "y" άξονα και στην τελευταία του "z" άξονα.



Σχήμα 9-10: Το πρόγραμμα Bray's Terminal λαμβάνει δεδομένα.

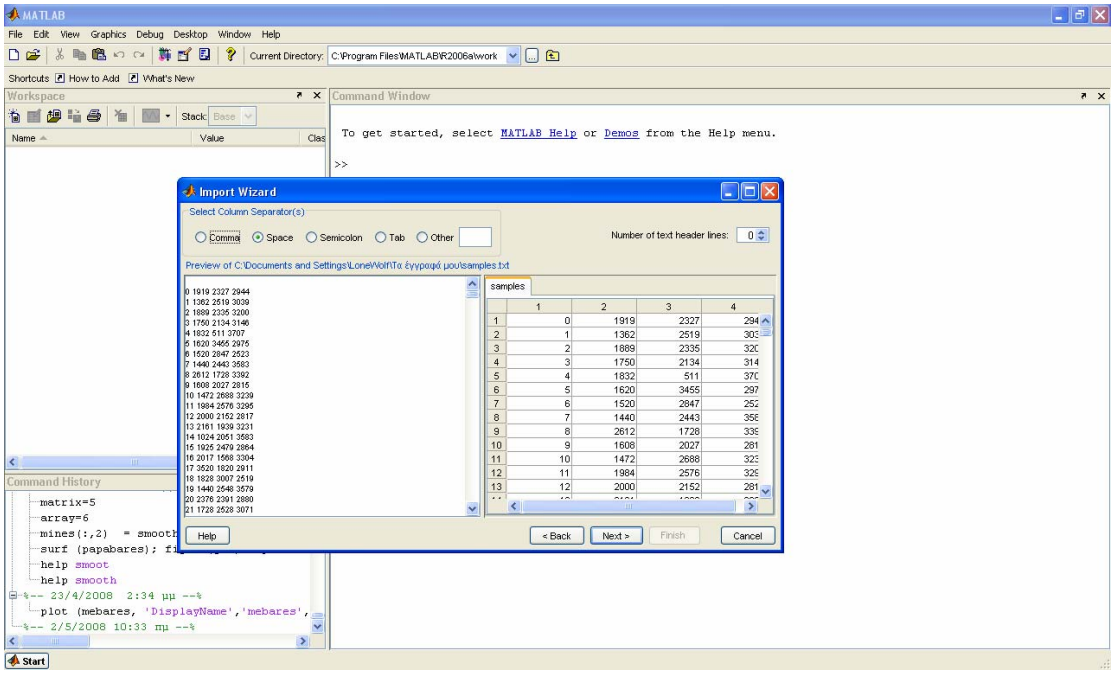


Σχήμα 9-11: Το αρχείο με τα δεδομένα που στάλθηκαν στο Bray's Terminal.

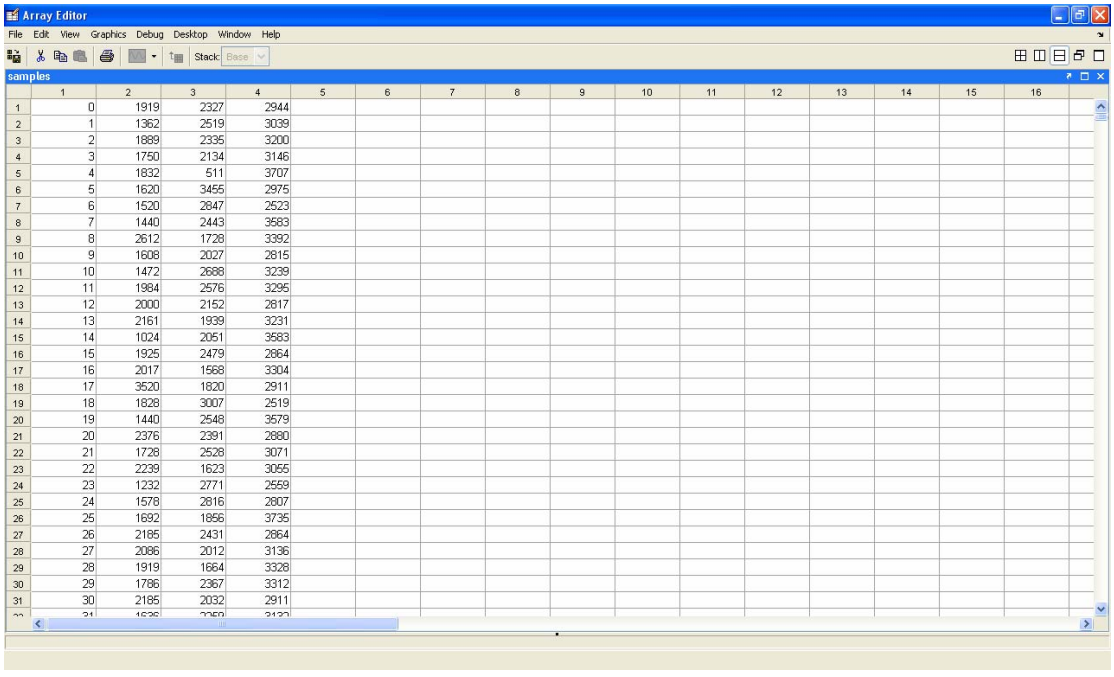
9.3 Επεξεργασία δεδομένων με το Matlab

9.3.1 Γενική προετοιμασία δεδομένων

Η επεξεργασία των δεδομένων γίνεται στο Matlab. Επειδή το πρόγραμμα "Bray's Terminal" έχει γράψει στο αρχείο διάφορα στοιχεία που δεν χρειάζεται το Matlab, όπως την ημερομηνία και την ώρα που πήρε τα δεδομένα, πρώτα ανοίγουμε το αρχείο, σβύνουμε τις περιττές πληροφορίες και το αποθηκεύουμε. Στην συνέχεια ανοίγουμε το Matlab και πάμε «File -> Import Data» και επιλέγουμε το αρχείο που το Bray's Terminal αποθήκευσε τα δεδομένα. Το Matlab τα αναγνωρίζει ως έναν πίνακα με 5000 γραμμές και 4 στήλες, όπως δηλαδή έχουν αποθηκευτεί. Πατάμε "Next" και μετά "Finish". Ο καινούριος πίνακας εμφανίζεται στο Workspace και έχει το όνομα του αρχείου. Αν κάνουμε plot τα δεδομένα που έχει, βλέπουμε ότι ο θόρυβος δεν επιτρέπει να βγάλουμε εύκολα συμπεράσματα. Γι΄αυτό τον λόγο περνάμε τα δεδομένα από ένα φίλτρο εξομάλυνσης. Πρώτα αντιγράφουμε τον πίνακα σε έναν καινούριο πχ. "array2=array1", και μετά εξομαλύνουμε κάθε στήλη με την εντολή smooth, πχ. "array2(:,i) = smooth(array1(:,i),20);" όπου i είναι η στήλη που θέλουμε i=2…4 και ο αριθμός 20 είναι τα δείγματα που λαμβάνει η μέθοδος κινητού μέσου για να κάνει την εξομάλυνση.

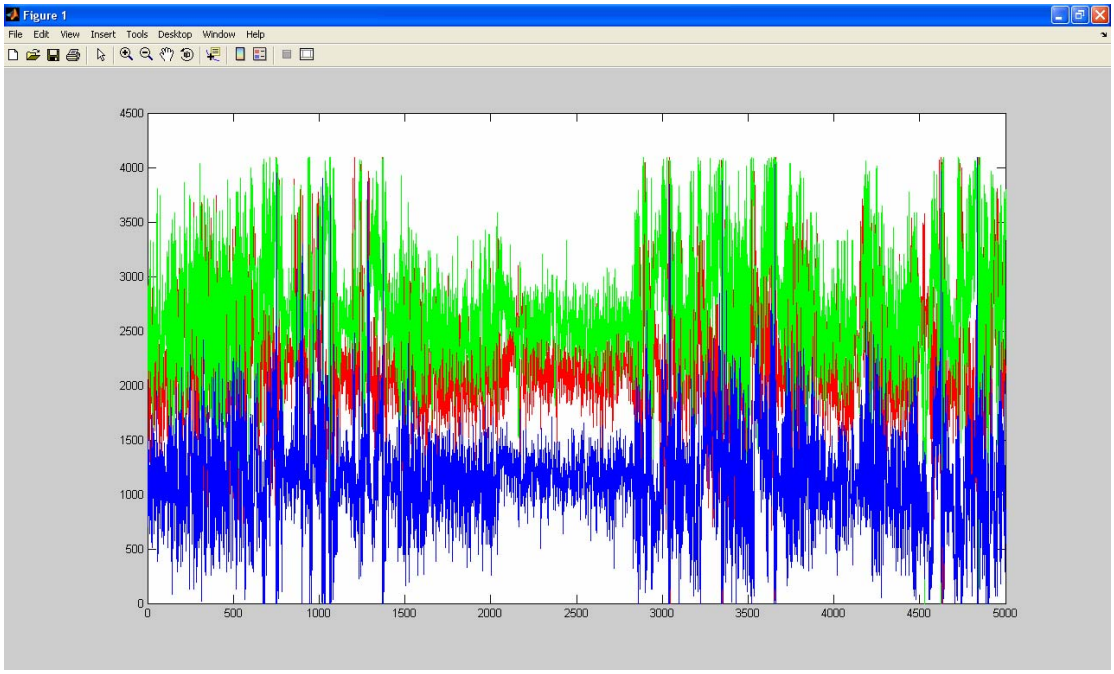


Σχήμα 9-12: Το πρόγραμμα Matlab διαβάζει τα δεδομένα από το αρχείο που έχουν αποθηκευτεί τα δείγματα.

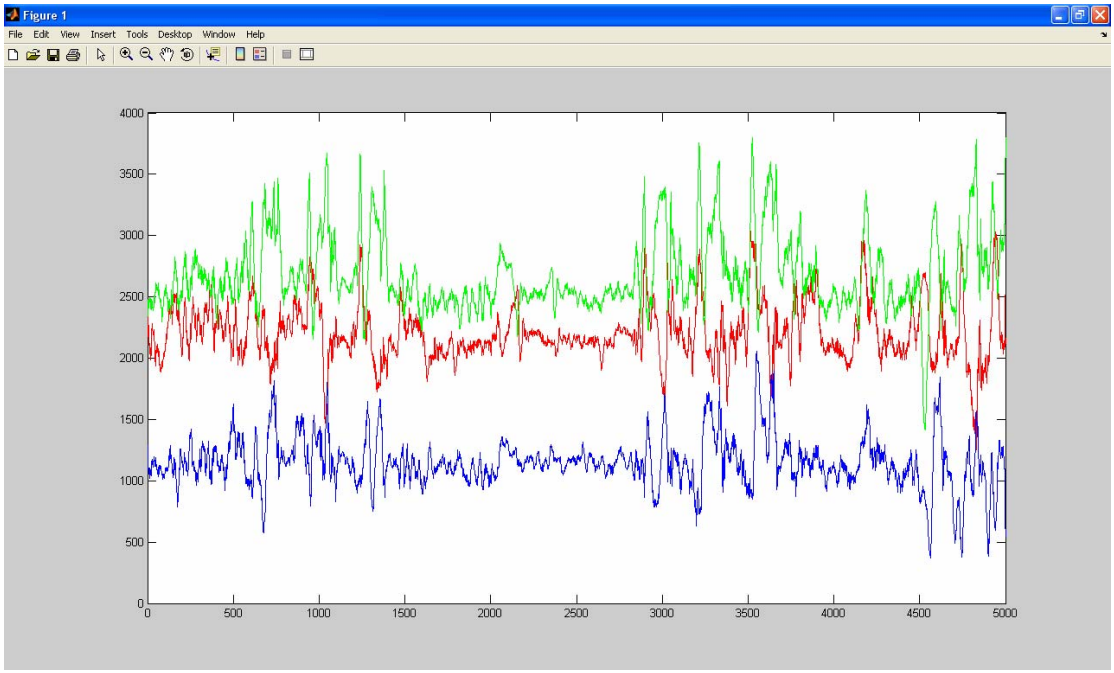


	1	2	3	4
1	0	1919	2327	2944
2	1	1362	2519	3039
3	2	1889	2335	3200
4	3	1750	2134	3146
5	4	1832	511	3707
6	5	1620	3455	2975
7	6	1520	2847	2523
8	7	1440	2443	3583
9	8	2612	1728	3392
10	9	1608	2027	2815
11	10	1472	2688	3239
12	11	1984	2576	3295
13	12	2000	2152	2817
14	13	2161	1939	3231
15	14	1024	2051	3583
16	15	1925	2479	2864
17	16	2017	1568	3304
18	17	3520	1820	2911
19	18	1828	3007	2519
20	19	1440	2548	3579
21	20	2376	2391	2880
22	21	1728	2528	3071
23	22	2239	1623	3055
24	23	1232	2771	2559
25	24	1578	2816	2807
26	25	1692	1856	3735
27	26	2185	2431	2864
28	27	2086	2012	3136
29	28	1919	1664	3328
30	29	1786	2367	3312
31	30	2185	2032	2911

Σχήμα 9-13: Ο πίνακας 4 στηλών και 5000 γραμμών που δημιούργησε το Matlab που περιέχει τα δεδομένα επιτάχυνσης.



Σχήμα 9-14: Γραφικές παραστάσεις των επιταχύνσεων x,y και z μαζί με θόρυβο.



Σχήμα 9-15: Γραφικές παραστάσεις των επιταχύνσεων x,y και z μετά από αφαίρεση του θορύβου.

9.3.2 Επεξεργασία δεδομένων με το αρχείο “gait1.m”

Στο Matlab έχουμε δημιουργήσει το αρχείο “gait1.m” το οποίο επεξεργάζεται τα σήματα μετατρέποντας την ψηφιακή τιμή τους σε τιμή επιτάχυνσης με μονάδα g, αφαιρεί κάποια dc συνιστώσα που μπορεί να οφείλεται στον προσανατολισμό του αισθητήρα και στην επιτάχυνση της βαρύτητας της γης, υπολογίζει τα σήματα ενέργειας, γινομένου και αθροίσματος των επιταχύνσεων που διευκολύνουν την ανάλυση, υπολογίζει την διακύμανση n σημείων του σήματος ενέργειας και τέλος δημιουργεί τις γραφικές παραστάσεις όλων των παραπάνω. Για να τρέξουμε το αρχείο γράφουμε “gait1(array2,100)”. Το πρώτο όρισμα είναι ο πίνακας με τις επιταχύνσεις που έχει όμως εξομαλυνθεί ενώ το δεύτερο όρισμα είναι η τιμές του δειγμάτων της διακύμανσης.

Ο πίνακας έχει αποθηκευμένα τα δεδομένα σε 4 στήλες και 5000 γραμμές. Σε κάθε γραμμή, η πρώτη στήλη είναι ο αριθμός του δείγματος, η δεύτερη είναι η ψηφιοποιημένη επιτάχυνση του x άξονα, ενώ η τρίτη και η τέταρτη είναι των y και z αξόνων αντίστοιχα για το συγκεκριμένο δείγμα.

```
function [en, pr, sm, varen] = gait1(data , n)

Limit=[3700:3950];

χ=((data(:,2))*0.80566/800);
y=((data(:,3))*0.80566/800)-2;
z=((data(:,4))*0.80566/800)-2.2;

y2 = y.^2;
z2 = z.^2;

en = y2 + z2;
pr = y.*z;
sm = y + z;

L = length(data(:,1));
i = 1;
while i <= L-n
    varen(i) = var(en(i:i+n));
    i = i+1;
end

figure
plot(Limit, y(Limit),'k');
title('Y acceleration')
figure
plot(Limit, z(Limit),'k');
title('Z acceleration');

figure
plot(Limit,en(Limit)/2666,'k');
title('Energy');
figure
plot(Limit,pr(Limit),'k');
title('Product')
```

```
figure
plot(Limit,sm(Limit),'k');
title('Sum')

figure
plot(Limit,varen(Limit),'k');
title(['Variance ',num2str(n)])
```

Ο πίνακας Limit = [3700:3950] καθορίζει ποια δείγματα θα παρουσιαστούν στις κυματομορφές εξόδου. Αυτό συμβαίνει επειδή θέλουμε να παρουσιάζουμε λίγα βήματα κάθε φορά για να βλέπουμε καλύτερα τα χαρακτηριστικά τους. Στην συνέχεια μετατρέπουμε τις ψηφιακές τιμές που έχουν τα δείγματα που λάβαμε, σε τιμές επιτάχυνσης g. Με αυτό τον τρόπο έχουμε καλύτερη επίβλεψη του μεγέθους που μετράμε. Η μετατροπή γίνεται ως εξής:

Επειδή ο ADC έχει 12bit ευαισθησίας και τάση αναφοράς 3.3V, για να βρούμε την τάση που μέτρησε, πολλαπλασιάζουμε την ψηφιακή τιμή με 0.80566mV [$(3.3/2^{12})$=3.3/4096 = 0.80566mV]. Επιπλέον επειδή ο αισθητήρας επιτάχυνσης είναι ρυθμισμένος σε ευαισθησία 800mV/g πρέπει να διαιρέσουμε με την τιμή αυτή. Το τελικό αποτέλεσμα είναι η επιτάχυνση σε g. Επίσης αφαιρούμε κάποια τιμή επιτάχυνσης από τις τιμές των y και z ώστε να εξαλείψουμε την τιμή της επιτάχυνσης της βαρύτητας και το dc offset που προσθέτει ο αισθητήρας για διευκόλυνση της ανάλυσης.

Οι μεταβλητές y2 και z2 είναι τα τετράγωνα των επιταχύνσεων των y και z αξόνων αντίστοιχα. Στην συνέχεια υπολογίζονται τα σήματα ενέργειας (en), γινομένου (pr) και αθροίσματος (sm) τα οποία είναι χρήσιμα στην ανάλυση των βημάτων.

Αμέσως μετά υπολογίζεται η διακύμανση n δειγμάτων χρησιμοποιώντας την συνάρτηση var. Τέλος σχεδιάζονται τα διαγράμματα με την συνάρτηση plot και δίνονται κατάλληλα ονόματα και τιμές στους άξονες. Παρατηρώντας τα διαγράμματα, μπορούμε και ανιχνεύουμε τα γεγονότα βηματισμού, τον αριθμό των βημάτων καθώς επίσης μπορούμε να συγκρίνουμε και να βρούμε ομοιότητες και διαφορές μεταξύ σημάτων διαφορετικών τύπων βηματισμού και ανθρώπων. Τα διαγράμματα παρουσιάζονται αναλυτικά στο κεφάλαιο 10.

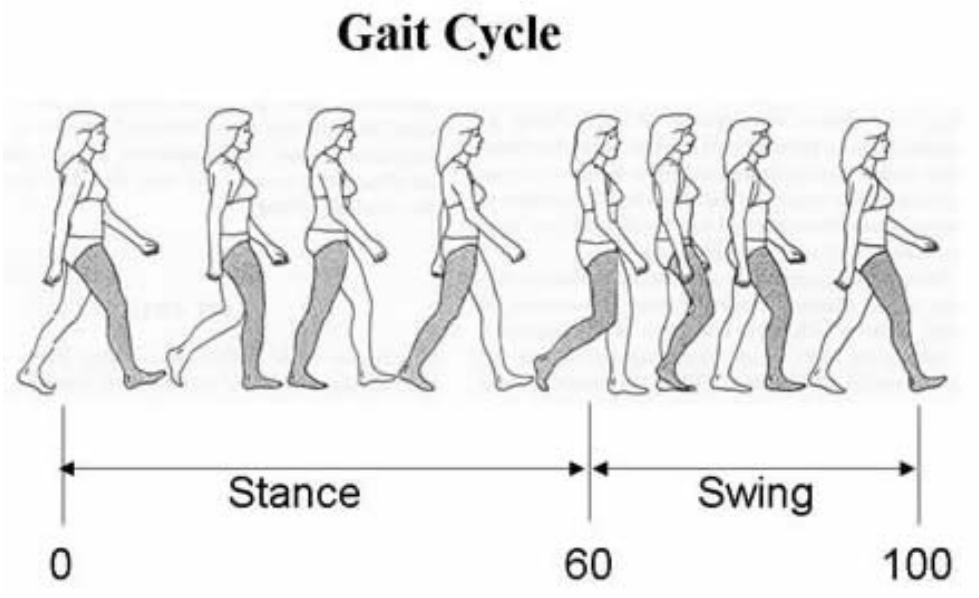
ΚΕΦΑΛΑΙΟ 10

Ανάλυση δεδομένων και πειραματικά αποτελέσματα

10.1 Ανάλυση βηματισμού

Στην ανάλυση βαδίσματος είναι σημαντικό να ξεχωρίζουμε τα διάφορα γεγονότα από τα οποία αποτελείται ο βηματισμός. Είναι σχετικά εύκολο να αναγνωρίζουμε τα γεγονότα αυτά παρατηρώντας τα σήματα που παίρνουμε από τον αισθητήρα επιτάχυνσης αλλά είναι χρήσιμη η ανάπτυξη κατάλληλων τεχνικών για την αυτοματοποιημένη εξακρίβωση των γεγονότων βηματισμού.

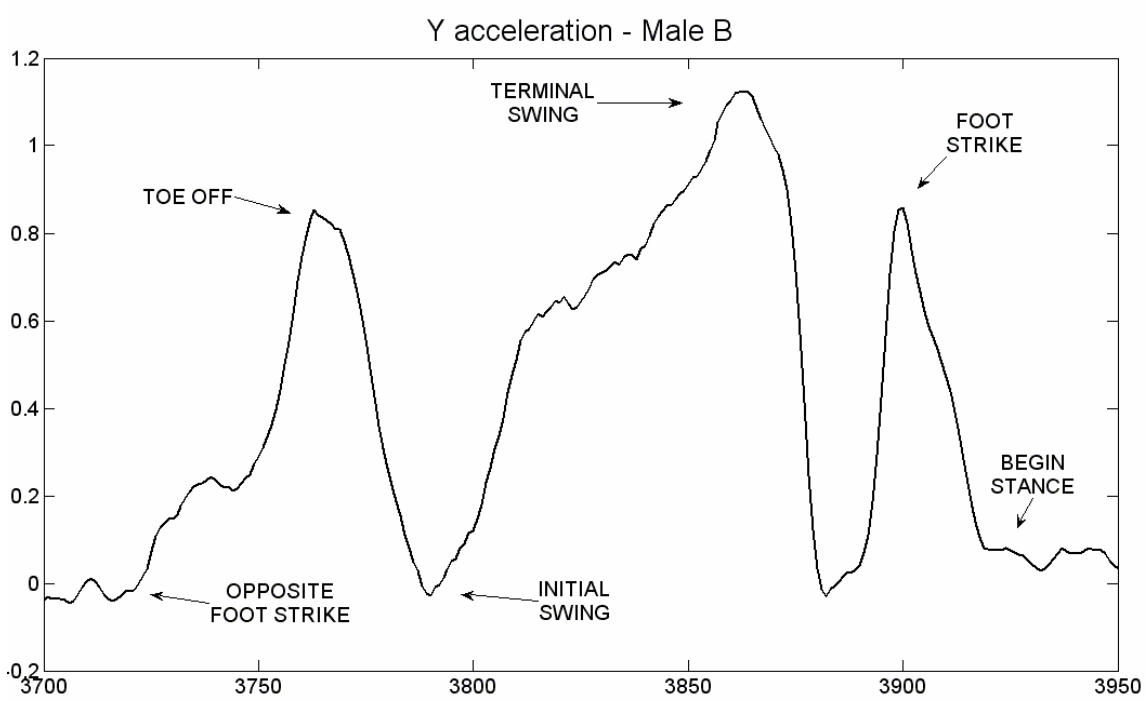
Στο σχήμα 10-1 φαίνεται ο κανονικός κύκλος βηματισμού του ανθρώπου. Στις μετρήσεις μας, η τοποθέτηση του επιταχυνσιομέτρου έγινε στο δεξί πόδι στο σημείο που φαίνεται στο σχήμα 10-2.



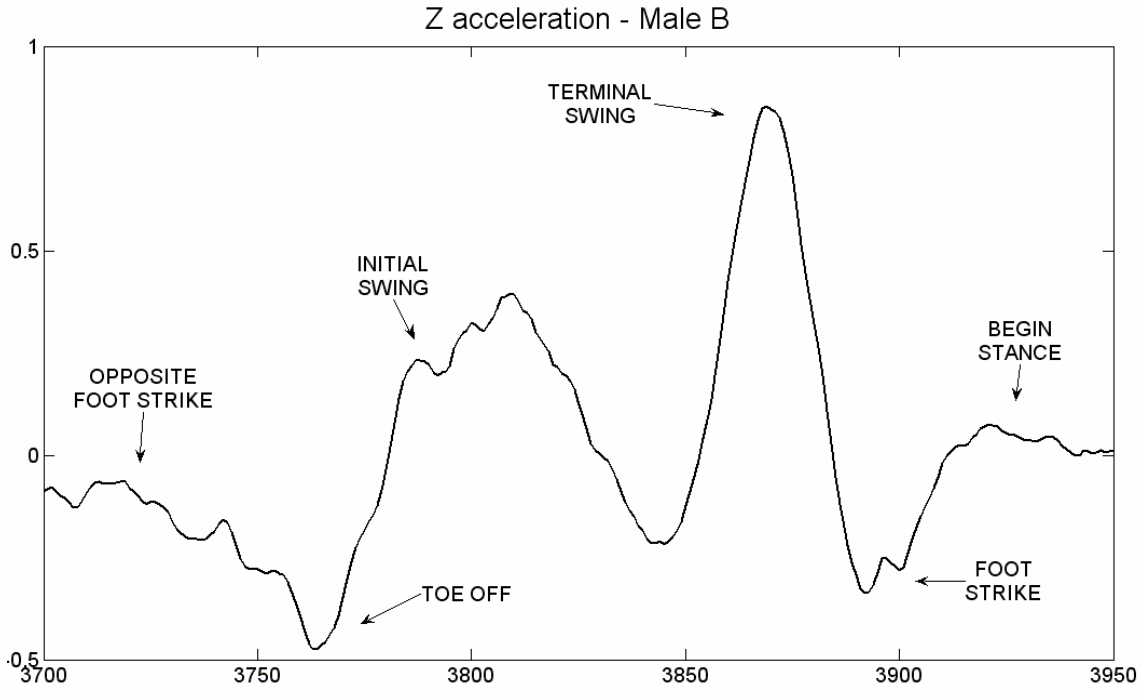
Σχήμα 10-1: Ο κύκλος βηματισμού του ανθρώπου.

Σχήμα 10-2: Οι άξονες των επιταχύνσεων.

Ο κύκλος βηματισμού ξεκινάει αμέσως μετά το χτύπημα του ποδιού στο έδαφος, συνήθως με τη φτέρνα, δημιουργώντας μια μεγάλη κορυφή επιτάχυνσης και ακολουθείται από ταλάντωση καθώς η σύγκρουση εξασθενίζεται από το παπούτσι και το σώμα του ανθρώπου. Το βάρος μεταφέρεται μπροστά και το αντίθετο πόδι αρχίζει να σηκώνεται από το έδαφος ξεκινώντας μια περίοδο στήριξης στο ένα πόδι, στην οποία το πόδι με το επιταχυνσιόμετρο είναι ακίνητο και η έξοδος του αισθητήρα σταθερή. Αν και το πόδι δεν επιταχύνεται εκείνη την περίοδο, το επιταχυνσιόμετρο θα μετράει την επιτάχυνση της βαρύτητας της γης. Καθώς το πόδι περιστρέφεται προς τα εμπρός, η επιτάχυνση που μετρίεται σε κάθε άξονα θα αλλάζει λίγο και ύστερα όλο και περισσότερο μετά το χτύπημα του αντίθετου ποδιού. Η περιστροφή του ποδιού επιταχύνεται μέχρι να σηκωθεί από το έδαφος και να ξεκινήσει η φάση αιώρησης. Η επιτάχυνση αυξάνεται από την αρχή της φάσης αιώρησης καθώς το πόδι αρχίζει να κινείται προς τα εμπρός. Η ταχύτητα εμπρόσθιας κίνησης του ποδιού αυξάνεται στη μέση της αιώρησης και μετά μειώνεται πριν το πόδι χτυπήσει στο έδαφος. Κατά την κρούση του ποδιού στο έδαφος δημιουργούνται απότομες αλλαγές στην επιτάχυνση.



Σχήμα 10-3: Η Y επιτάχυνση σε ένα κύκλο βηματισμού. Φαίνονται και τα διάφορα γεγονότα βηματισμού.



Σχήμα 10-4: Η Z επιτάχυνση σε ένα κύκλο βηματισμού. Φαίνονται και τα διάφορα γεγονότα βηματισμού.

10.2 Χρήσιμες μετατροπές σήματος

Η αναγνώριση των φάσεων του βηματισμού μπορεί να γίνει εύκολα με το μάτι αλλά η αυτοματοποίηση της αναγνώρισης για την αξιόπιστη επεξεργασία των διαφορετικών σημάτων που προκύπτουν από διαφορετικούς τρόπους βηματισμού είναι πιο δύσκολη. Η επεξεργασία των σημάτων πρέπει να μπορεί να ενισχύει τα γεγονότα που μας ενδιαφέρουν και να μειώνει τα γεγονότα που μας είναι αδιάφορα. Γι' αυτό το λόγο δημιουργούμε τα σήματα ενέργειας, γινομένου και αθροίσματος.

10.2.1 Ενέργεια επιτάχυνσης

Η ενέργεια του σήματος επιτάχυνσης είναι το άθροισμα των τετραγώνων των επιταχύνσεων που μετριούνται στους ορθογώνιους άξονες Y και Z.

$$Energy = Y^2 + Z^2$$

Η ενέργεια είναι χρήσιμη γιατί ενισχύει περιόδους μεγάλης δυναμικής, εξουδετερώνει περιόδους χαμηλής δυναμικής και δημιουργεί ένα σήμα που είναι πάντα θετικό. Η ακολουθία των γεγονότων στο σήμα ενεργείας είναι πολύ καλά καθορισμένη. Νωρίς κατά τη διάρκεια της φάσης στάσης, όταν το πόδι είναι σταθερό, το επιταχυνσιόμετρο μετρά κάποια σταθερή επιτάχυνση. Όταν το πόδι σηκώνεται από το έδαφος δημιουργεί υψηλότερη και στενότερη κορυφή από εκείνη του αρχικού σήματος. Ένα χρήσιμο και ανιχνεύσιμο γεγονός συμβαίνει στην αρχική φάση αιώρησης όταν η ενέργεια επιτάχυνσης είναι λίγο μικρότερη από την φάση στάσης ακριβώς πριν από μια συγκρατημένη αύξηση στην τελική φάση αιώρησης. Η τελευταία περίοδος του κύκλου βηματισμού έχει ταλαντώσεις που τελειώνουν με μια μεγάλη κορυφή κατά τη φάση χτυπήματος του ποδιού στο έδαφος. Τα κύρια χαρακτηριστικά που μπορούμε να εξάγουμε από αυτό το σήμα είναι η φάση που σηκώνεται το πόδι από το έδαφος, η αρχική φάση αιώρησης και το χτύπημα του ποδιού στο έδαφος.

10.2.2 Γινόμενο επιτάχυνσης

Το δεύτερο χρήσιμο σήμα που μπορεί να κατασκευαστεί από τα δεδομένα επιτάχυνσης που έχουμε πάρει είναι το γινόμενο επιτάχυνσης. Εκεί που το σήμα ενέργειας επιτάχυνσης μας βοηθάει να διαχωρίσουμε σχετικά πλάτη των δυναμικών βηματισμού, το γινόμενο των ορθογώνιων αξόνων Y και Z αναγνωρίζει περιόδους που οι επιταχύνσεις έχουν αντίθετα πρόσημα.

$$\mathrm{Pr}oduct = Y \cdot Z$$

Στο σήμα επιτάχυνσης μπορούμε να ανιχνεύσουμε την φάση που σηκώνεται το πόδι από το έδαφος που είναι ένα μεγάλο ελάχιστο, με το αμέσως επόμενο μέγιστο μπορούμε να ανιχνεύσουμε την αρχική φάση αιώρησης, το δεύτερο μέγιστο αποτελεί την τελική φάση αιώρησης και το επόμενο ελάχιστο είναι το χτύπημα του ποδιού στο έδαφος.

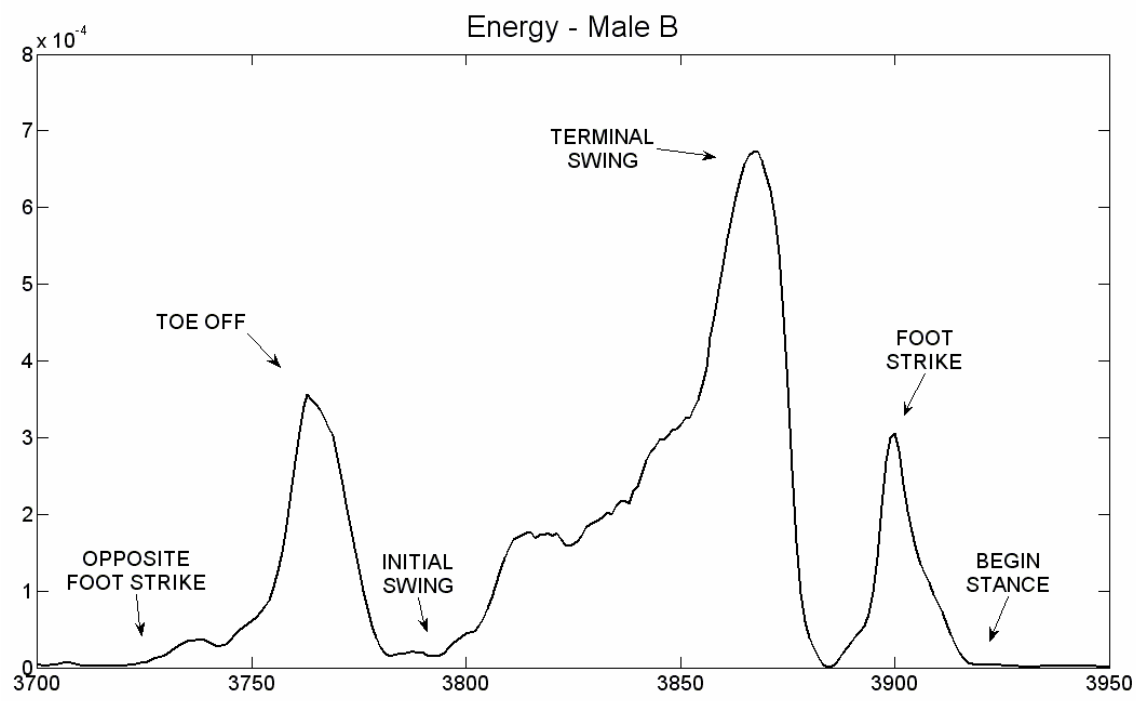
Κατά τη τελική φάση αιώρησης, το σήμα γινομένου ξεχωρίζει την τελική φάση αιώρησης με ένα μέγιστο. Στο σήμα ενεργείας το γεγονός αυτό είναι πιο δύσκολο να διακριθεί επειδή βρίσκεται κοντά σε κορυφή παρομοίου μεγέθους.

10.2.3 Άθροισμα επιτάχυνσης

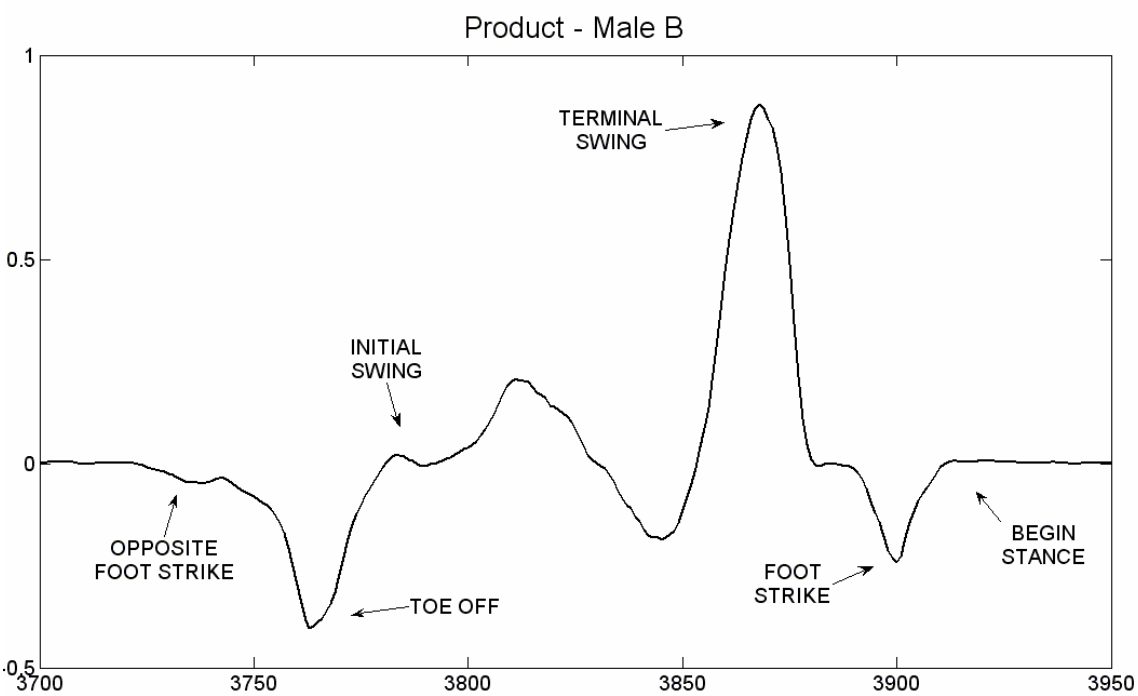
Το τρίτο χρήσιμο σήμα είναι το άθροισμα των επιταχύνσεων των Y και Z αξόνων.

$$Sum = Y + Z$$

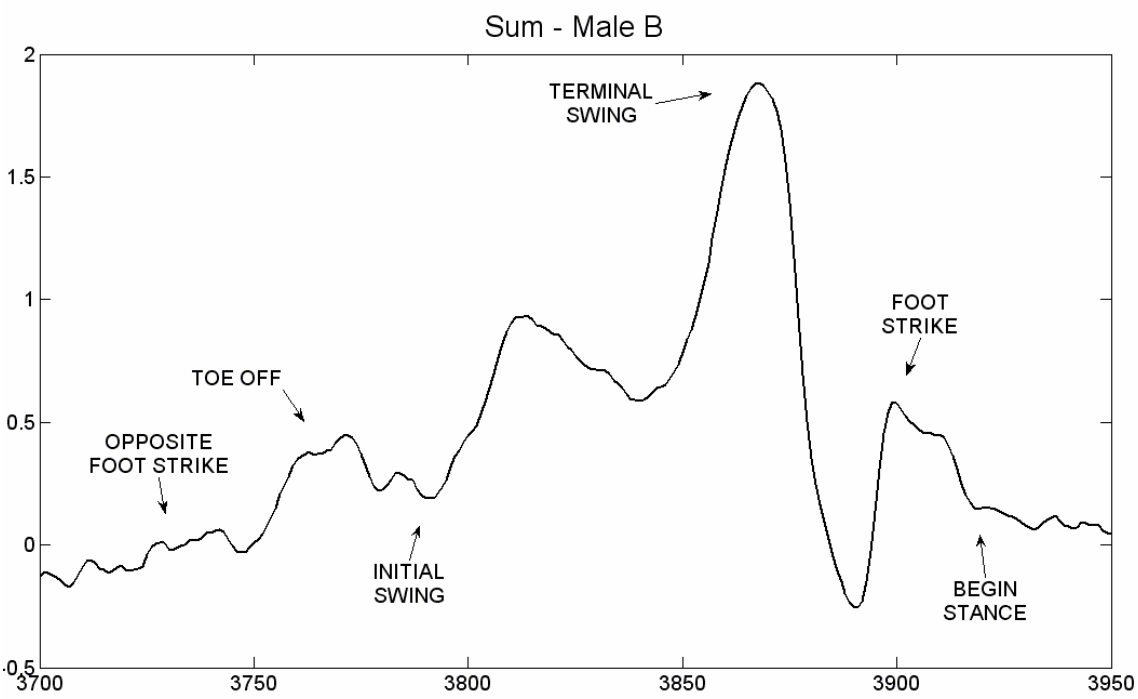
Το άθροισμα επιτάχυνσης αναδεικνύει τη φάση τέλους της αιώρησης. Συγκεκριμένα δημιουργεί ένα μέγιστο στο τέλος της αιώρησης του ποδιού. Ομοίως ανιχνεύεται έυκολα και το χτύπημα του ποδιού καθώς εμφανίζεται ως απότομη αύξηση μετά από ένα ελάχιστο. Επειδή το άθροισμα είναι εξίσωση πρώτου βαθμού, υπάρχει μεγαλύτερη διακύμανση στο σήμα κατά τη φάση στάσης κάνοντάς το λιγότερο κατάλληλο για την αναγνώριση αυτών των γεγονότων.



Σχήμα 10-5: Η ενέργεια επιτάχυνσης.



Σχήμα 10-6: Το γινόμενο επιτάχυνσης.



Σχήμα 10-7: Το άθροισμα επιτάχυνσης.

10.3 Ανίχνευση γεγονότων

Έχοντας μετατρέψει τα σήματα με τις επιταχύνσεις που πήραμε με το επιταχυνσιόμετρο σε σήματα που τονίζουν διάφορα γεγονότα βηματισμού είναι σημαντικό να δημιουργήσουμε κάποια κριτήρια για να τα ανιχνεύουμε. Σε μια πρακτική εφαρμογή είναι χρήσιμο να ανιχνεύουμε τα γεγονότα όσο πιο γρήγορα γίνεται για να μεγιστοποιήσουμε την απόκριση της μέτρησης του μήκους βηματισμού. Αυτό σημαίνει ότι ο αλγόριθμος αναγνώρισης πρέπει να δουλεύει με έναν ελάχιστο αριθμό δειγμάτων. Επίσης πρέπει να είναι ικανός να δουλεύει σε διάφορες ταχύτητες και τρόπους βηματισμού.

Οι μετρήσεις μπορούν να αρχίσουν με τον χρήστη να περπατάει, συνεπώς πρέπει πρώτα να προσδιορίσουμε σε ποια φάση του κύκλου βηματισμού βρίσκεται ο χρήστης τη στιγμή που αρχίζει η μέτρηση. Αυτό μπορεί να γίνει ψάχνοντας στα ένα με δύο πρώτα δευτερόλεπτα δεδομένων για κάποιο αναγνωρίσιμο γεγονός όπως η φάση στάσης και μετά γυρίζοντας πίσω στην αρχή των μετρήσεων. Μόλις ξεχωρίσουμε την τρέχουσα φάση είναι δυνατό να περιμένουμε το επόμενο γεγονός και να ελέγχουμε για τις αναμενόμενες τιμές. Με αυτό τον τρόπο η ανίχνευση απλοποιείται.

10.4 Κατώφλια πλάτους

Ο απλούστερος τρόπος για αναγνώριση γεγονότων είναι να ελέγχουμε για τα σημεία στα οποία το σήμα περνάει από κάποια τιμή κατωφλιού. Για παράδειγμα μπορούμε να πούμε ότι αν στα τελευταία πέντε δείγματα του σήματος ενέργειας οι τιμές ήταν πολύ κοντά στο $1g^2$ τότε βρισκόμαστε στη φάση στάσης. Επίσης μπορούμε να ανιχνεύσουμε φάσεις χτυπήματος ποδιού στο έδαφος, ψάχνοντας για τιμές ενέργειας μεγαλύτερες από $10g^2$. Ωστόσο χρησιμοποιώντας μόνο την τιμή του πλάτους του σήματος για την αναγνώριση, υπάρχει το ρίσκο ότι μια απρόβλεπτη αλλαγή στις καταστάσεις μπορεί να προκαλέσει λανθασμένη ή ασυνεχή αναγνώριση γεγονότων. Ένα μικρό λάθος στην βαθμονόμηση του αισθητήρα μπορεί να οδηγήσει το σήμα ενέργειας ώστε να μην φτάνει ποτέ την τιμή $1g^2$. Παρομοίως, μια τιμή κατωφλιού που έχει τεθεί για μέσο περπάτημα μπορεί να μην ανιχνεύσει τα χτυπήματα ποδιού ενός ανθρώπου που περπατάει σιγά, ενώ ένα άνθρωπος που περπατάει δυνατά μπορεί να περάσει την τιμή του κατωφλιού πολλές φορές σε ένα βηματισμό. Συχνά η απόφαση για το αν κάποιο γεγονός έγινε, εξαρτάται από το αν η τιμή μιας μεταβλητής είναι μικρότερη ή μεγαλύτερη από κάποιο επιλεγμένο κατώφλι. Είναι σημαντικό η μεταβλητή που συγκρίνεται να είναι ευαίσθητη μόνο στα δυναμικά στοιχεία του σήματος που οφείλονται στο περπάτημα.

10.5 Κατώφλι διακύμανσης

Η διακύμανση (variance) s_n^2, n δειγμάτων δεδομένων είναι :

$$S_{n_j}{}^2 = \frac{1}{n} \sum_{i=j}^{i=j+n} \left(x_i - \overline{x}_j \right)^2$$

Όπου $\overline{x}_j$ είναι η μέση τιμή n δειγμάτων.

$$\overline{x}_j = \frac{1}{n} \sum_{i=j}^{i=j+n} x_i$$

Η διακύμανση μπορεί να χρησιμοποιηθεί για την ανίχνευση γεγονότων όπου το σήμα αλλάζει απότομα, όπως το χτύπημα της φτέρνας και η αρχή της φάσης στάσης. Επειδή η διακύμανση μετράει την σχετική σταθερότητα του σήματος, δεν επηρεάζεται από την πόλωση και τις θερμοκρασιακές αλλαγές του αισθητήρα. Συνεπώς οι ανιχνεύσεις κατωφλιού χρησιμοποιώντας την διακύμανση δουλεύουν πιο αξιόπιστα.

Η επιλογή του αριθμού των δειγμάτων που παίρνουν μέρος στον υπολογισμό της διακύμανσης επηρεάζει την αναγνώριση γεγονότων. Η αρχή της φάσης στάσης βρίσκεται εκεί που η τιμή της διακύμανσης πέφτει απότομα από ένα υψηλό μέγιστο σε μια χαμηλή σταθερή τιμή.

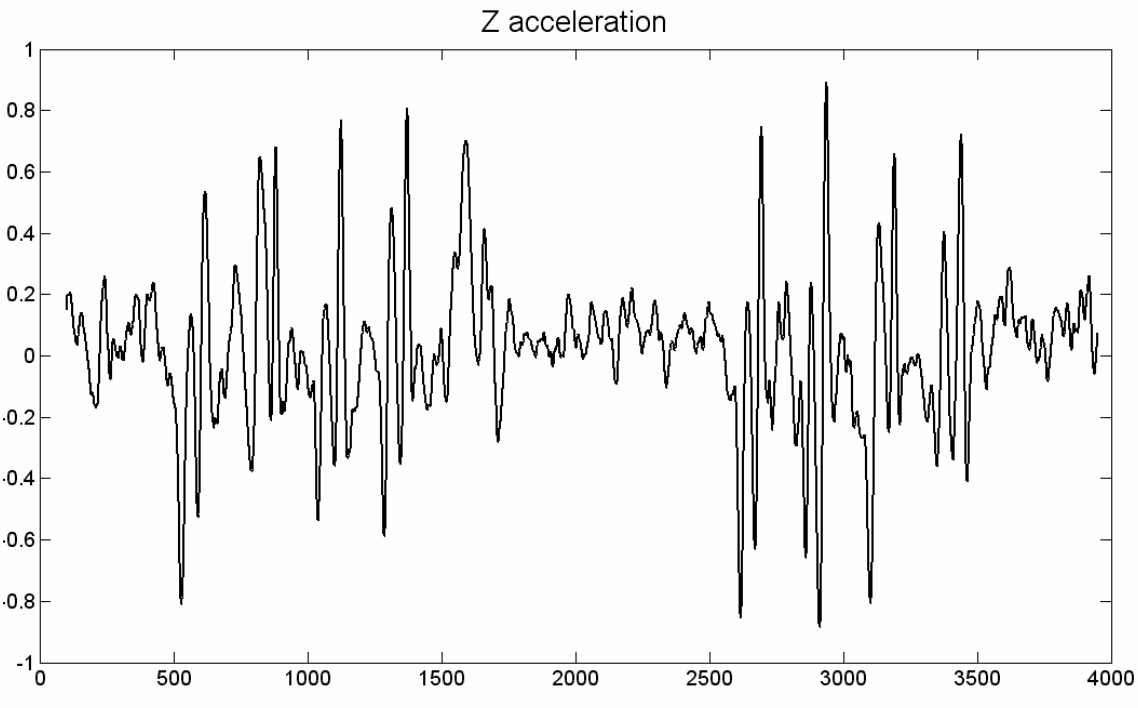
Η διακύμανση που προκύπτει από λίγα δείγματα αποκρίνεται σε γρήγορες αλλαγές στην ενέργεια με υψώματα μεγάλου πλάτους. Υπάρχει μικρή χρονική καθυστέρηση μεταξύ του γεγονότος βηματισμού και της αναγνώρισής του, αλλά εμφανίζονται και άλλα γεγονότα που σχετίζονται με την αρχική αιώρηση. Αυτό οφείλεται επειδή η περίοδος αργών αλλαγών στην ενέργεια στην φάση αρχικής αιώρησης δεν μπορεί να διαχωριστεί από την περίοδο της μη αλλαγής στην ενέργεια στην φάση στάσης μόνο με τον υπολόγισμό της διακύμανσης με τόσο μικρό αριθμό δειγμάτων.

Χρησιμοποιώντας έναν μεγαλύτερο αριθμό δειγμάτων η διακύμανση που προκύπτει είναι πιο ευαίσθητη σε πιο αργές μεταβολές στην ενέργεια. Με αυτόν τον τρόπο προκύπτουν πιο εύκολα οι αλλαγές στην φάση στάσης. Ωστόσο οι ανίχνευση καθυστερεί περισσότερο καθώς χρειάζεται μεγαλύτερος αριθμός δειγμάτων στους υπολογισμούς.

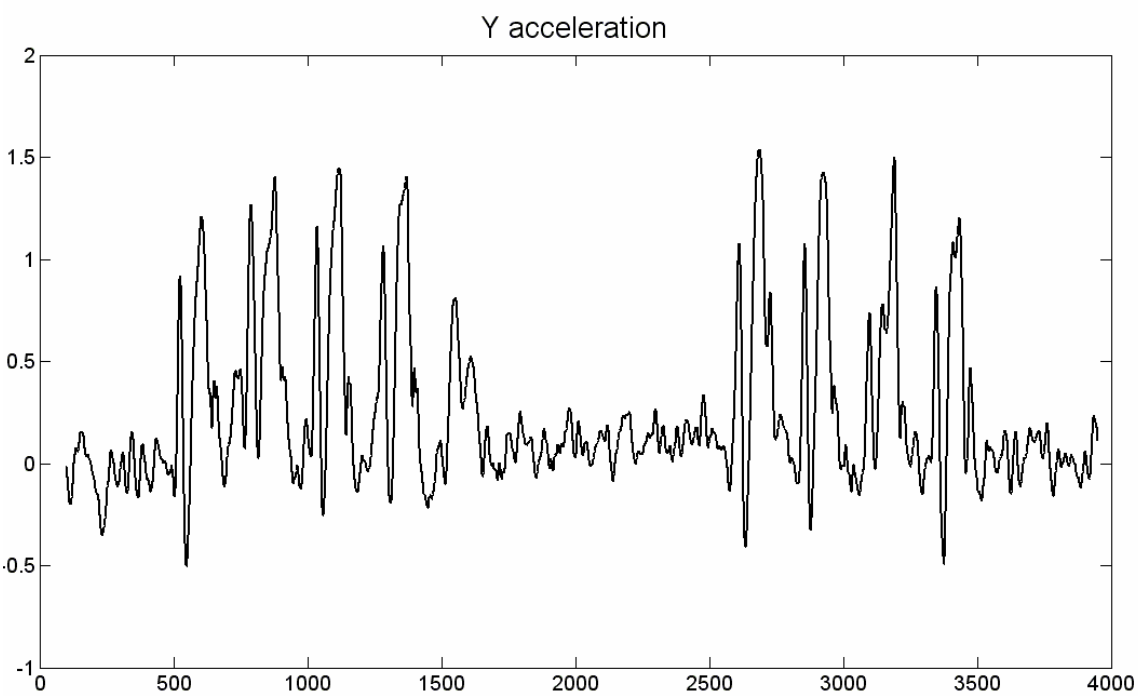
Αν η διακύμανση ενός σήματος που έχει υποστεί μετατροπή προκειται να χρησιμοποιηθεί για αναγνώριση γεγονότων βηματισμού, ο αριθμός των δειγμάτων n πρέπει να επιλεγεί κατάλληλα ώστε η διακύμανση να είναι αρκετά ευαίσθητη σε αργές μεταβολές και να αποκρίνεται ικανοποιητικά σε γρήγορες μεταβολές. Με κατάλληλες επιλογές κατωφλιού βρέθηκε ότι όταν ο αριθμός δειγμάτων είναι μεταξύ 70 και 120 τα αποτελέσματα ανίχνευσης ήταν

ικανοποιητικά για ένα μεγάλο εύρος τύπων περπατήματος ενώ για άλλες τιμές μικρότερες και μεγαλύτερες ο διαχωρισμός των βημάτων δεν ήταν δυνατός.

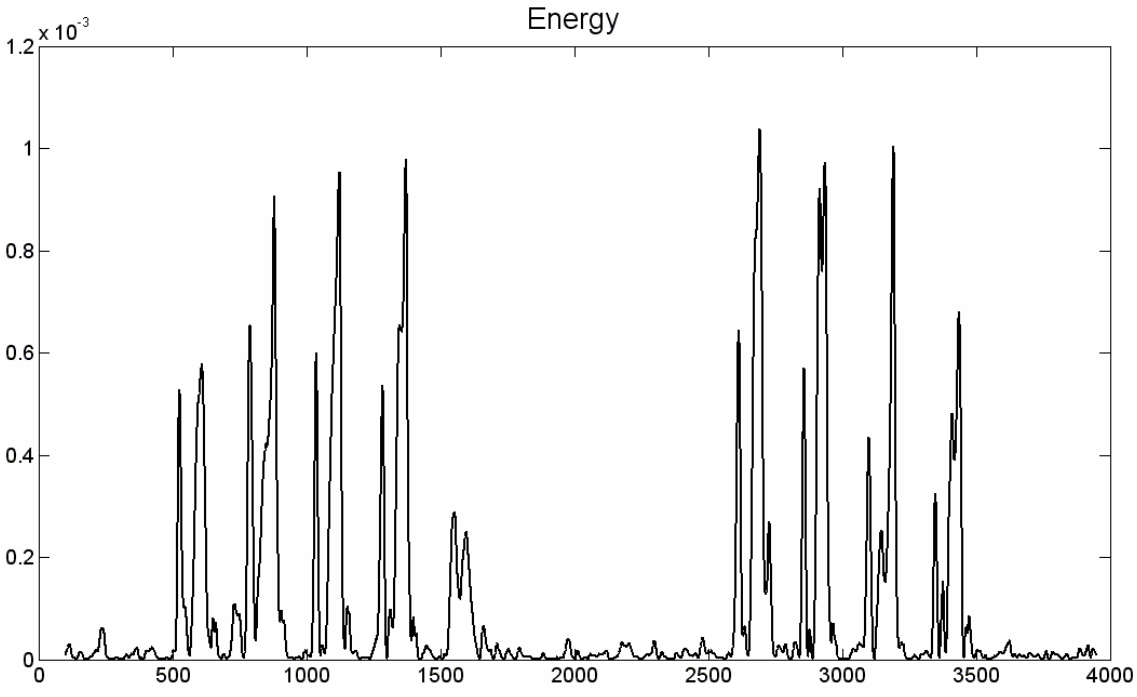
Παρακάτω εμφανίζονται τα διαγράμματα των επιταχύνσεων Z και Y, της ενέργειας, του γινομένου και της διακύμανσης 100 σημείων, ενός σήματος οκτώ βημάτων. Παρατηρούμε ότι στα σήματα εκτός της διακύμανσης είναι πολύ δύσκολο να ορίσουμε κάποιο κατώφλι για την αναγνώριση των βημάτων, λόγω γειτονικών κορυφών σε κάθε βήμα, λόγω διαφορετικής ασκούμενης δύναμης στον βηματισμό που δημιουργεί διαφορετικά μεγέθη κορυφών και λόγω του θορύβου. Ωστόσο στην διακύμανση το σήμα είναι ξεκάθαρο και κάθε βήμα μπορεί να ανιχνευτεί θέτοντας ένα κατώφλι ή ανιχνεύοντας απότομες αλλαγές στο σήμα. Επίσης αξίζει να παρατηρηθεί ότι μετά το τέταρτο βήμα ο χρήστης συνάντησε κάποιο εμπόδιο και έκανε στροφή πριν ξαναρχίσει να περπατάει. Αυτό φαίνεται στις Y και Z επιταχύνσεις αλλά στα υπόλοιπα σήματα έχει μειωθεί αισθητά ανώ στα σήματα διακυμάνσεων δεν αναγνωρίζεται σαν βήμα. Αυτό σημαίνει ότι οι μετατροπές σήματος χρησιμεύουν και για να μηδενίζουν στοιχεία που δεν έχουν σχέση με το βηματισμό αν και το πλάτος τους είναι συγκρίσιμο με τα πλάτος των βημάτων.



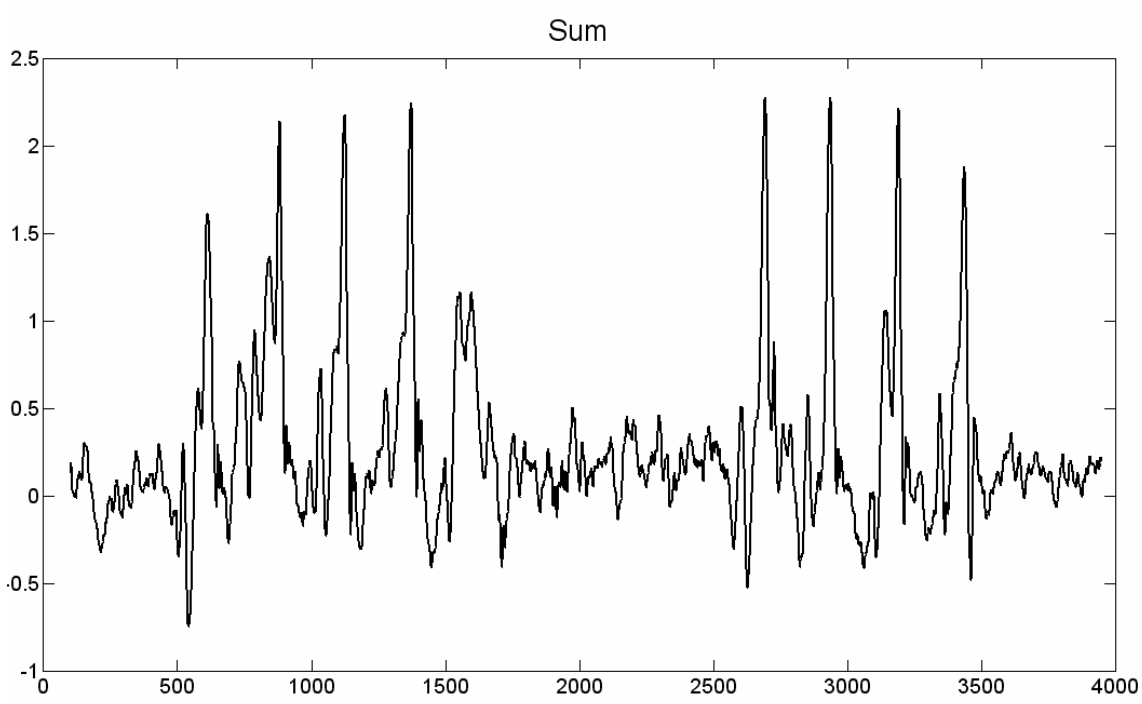
Σχήμα 10-8: Στην Z επιτάχυνση δεν μπορούμε εύκολα να θέσουμε ένα κατώφλι για τον διαχωρισμό των βημάτων.



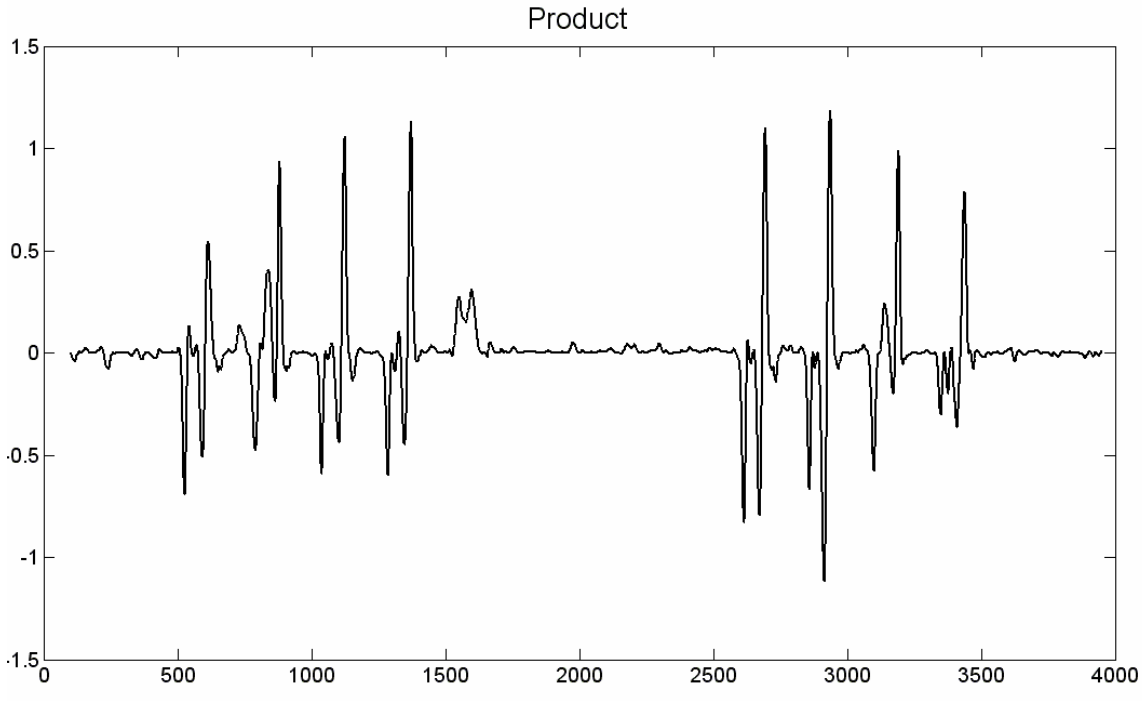
Σχήμα 10-9: Στην Y επιτάχυνση δεν μπορούμε εύκολα να θέσουμε ένα κατώφλι για τον διαχωρισμό των βημάτων.



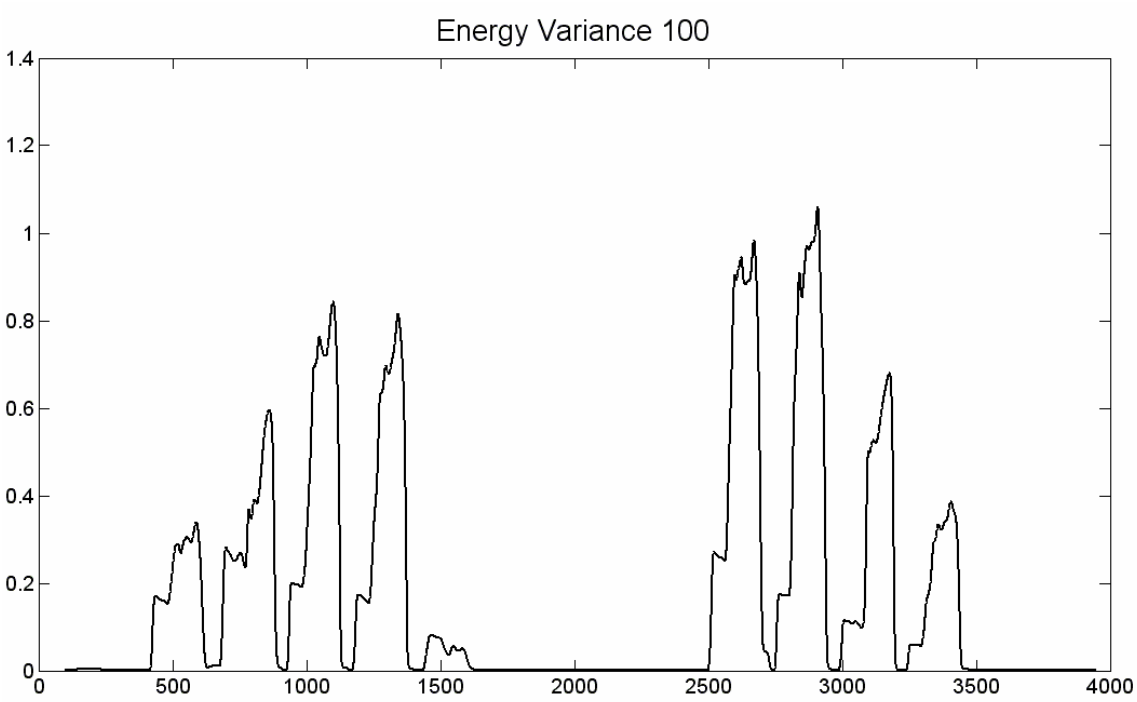
Σχήμα 10-10: Στην ενέργεια επιτάχυνσης δεν μπορούμε εύκολα να θέσουμε ένα κατώφλι για τον διαχωρισμό των βημάτων.



Σχήμα 10-11: Στο άθροισμα επιτάχυνσης δεν μπορούμε εύκολα να θέσουμε ένα κατώφλι για τον διαχωρισμό των βημάτων.

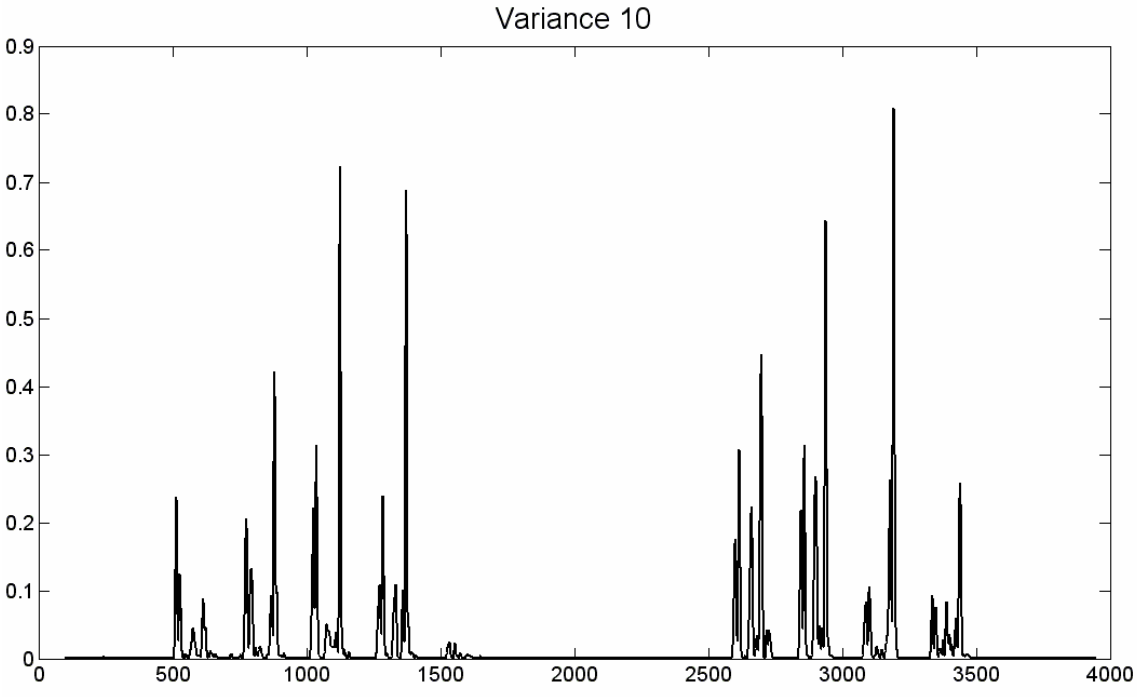


Σχήμα 10-12: Στο γινόμενο επιτάχυνσης δεν μπορούμε εύκολα να θέσουμε ένα κατώφλι για τον διαχωρισμό των βημάτων.

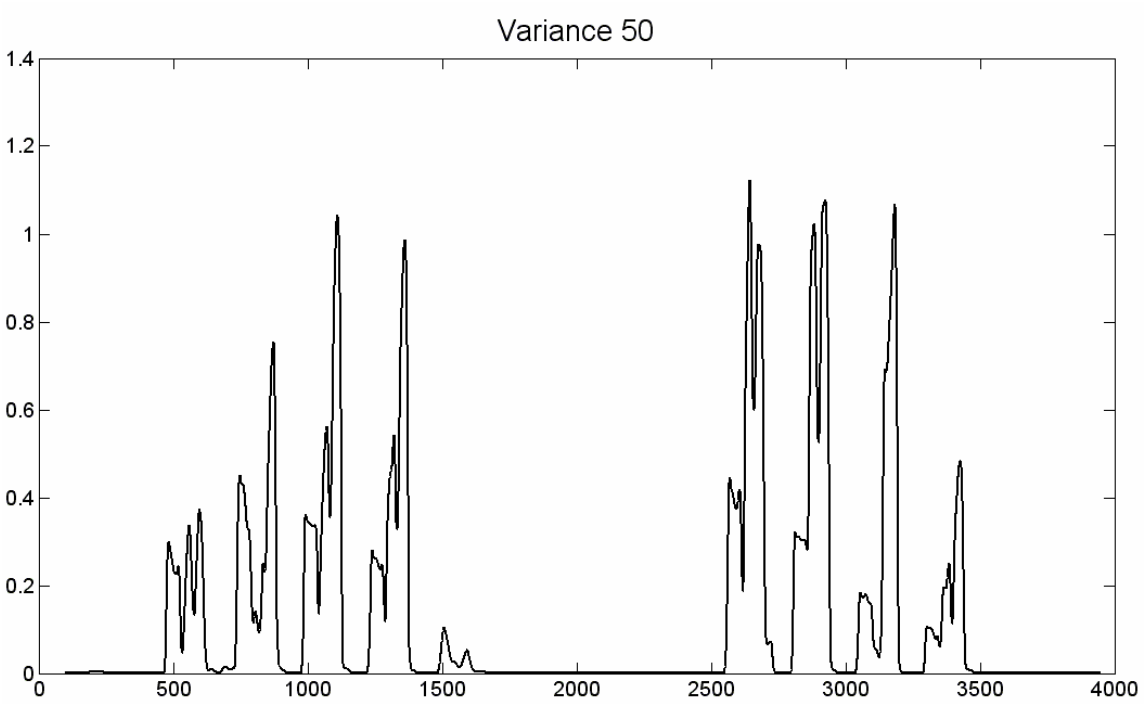


Σχήμα 10-13: Στη διακύμανση επιτάχυνσης μπορούμε εύκολα να θέσουμε ένα κατώφλι για τον διαχωρισμό των βημάτων. Εδώ εμφανίζονται έξι βήματα και μπορούμε να τα ξεχωρίσουμε με ένα κατώφλι 0.6.

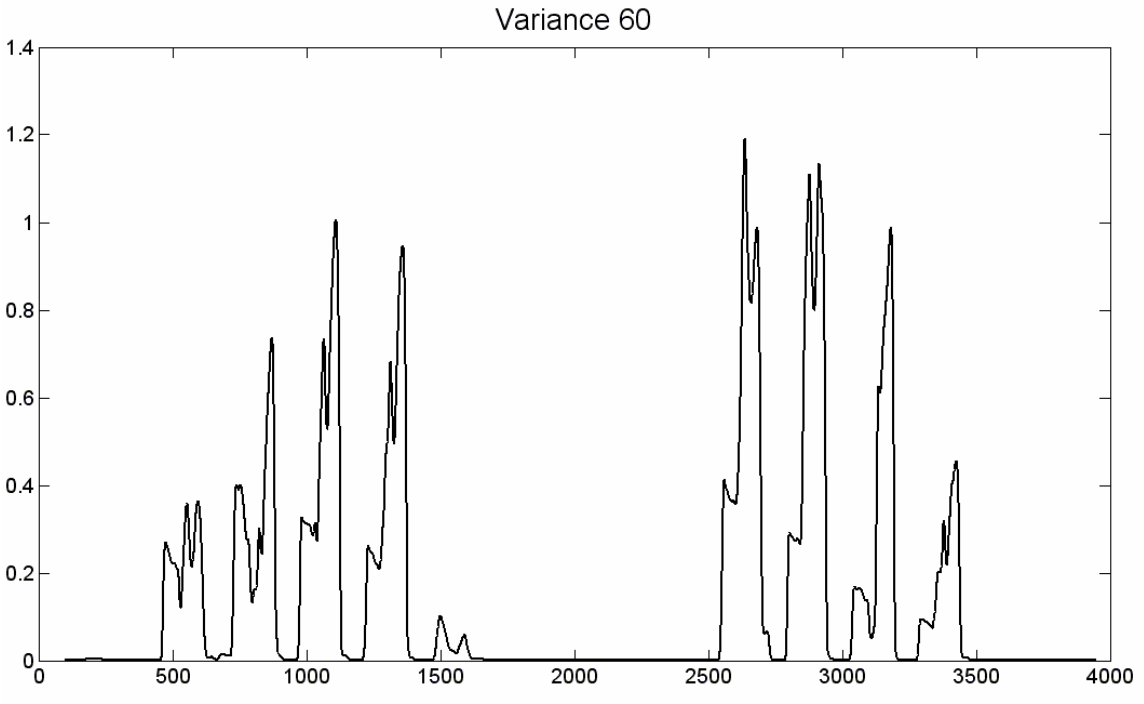
Παρακάτω παρουσιάζονται οι διακυμάνσεις για διαφορετικές τιμές δειγμάτων.



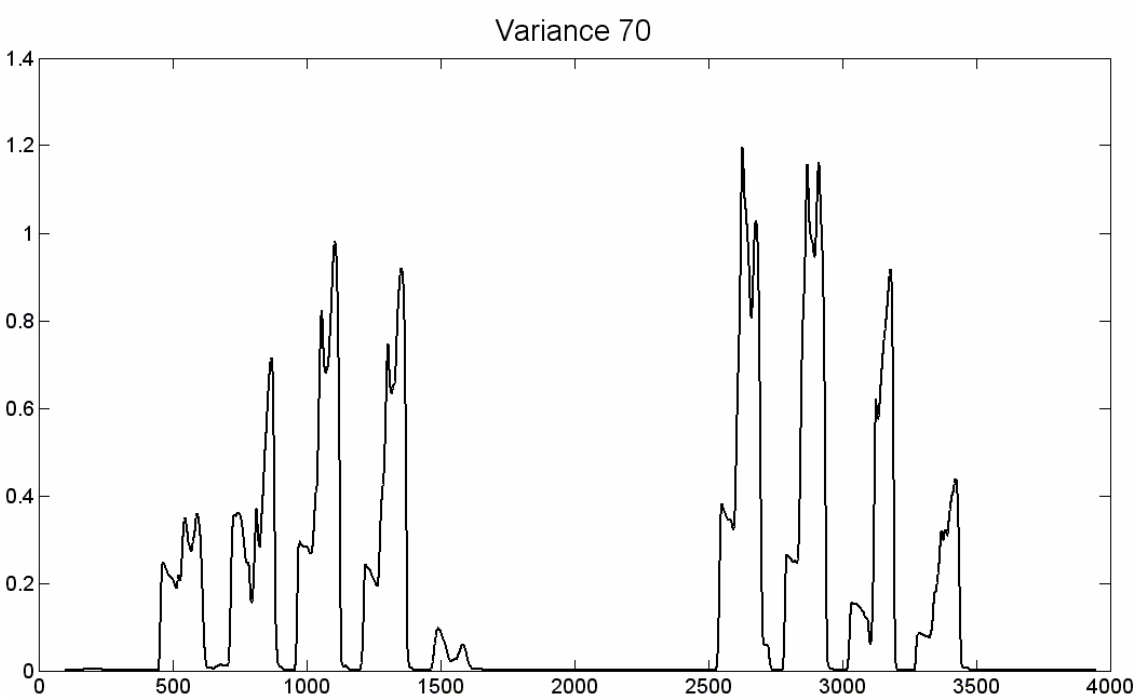
Σχήμα 10-14: Η διακύμανση με μικρό αριθμό δειγμάτων 10 δεν εξυπηρετεί για τον διαχωρισμό βημάτων.



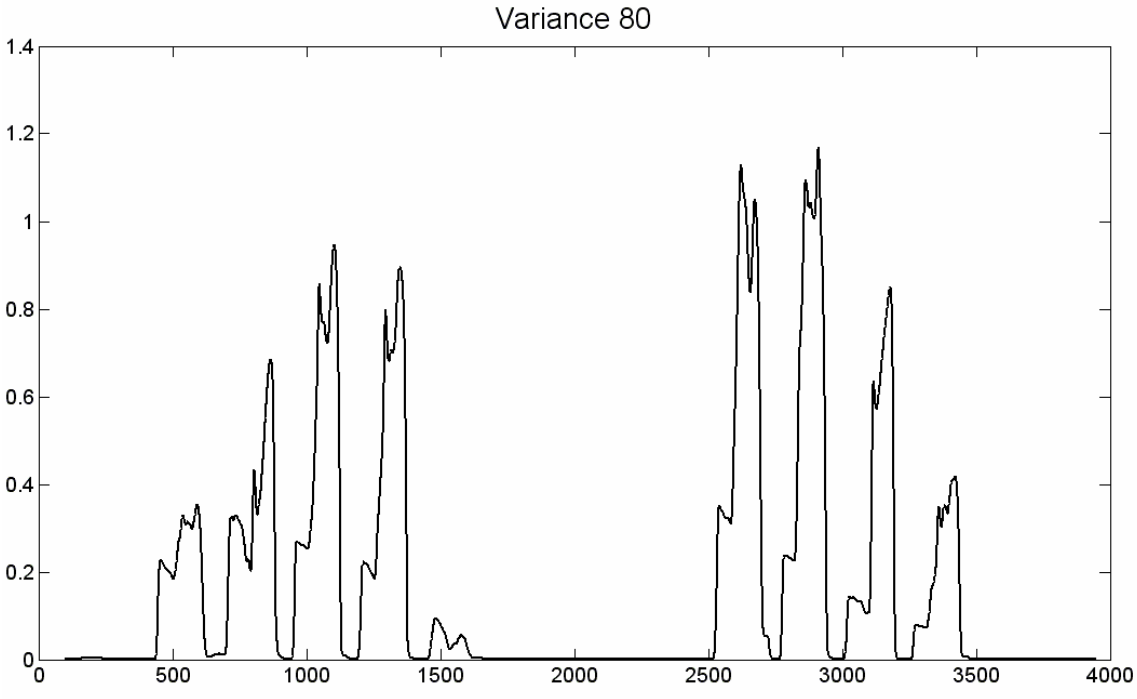
Σχήμα 10-15: Η διακύμανση με αριθμό δειγμάτων 50 αρχίζει να εμφανίζει κάποια ευκολία στον προσδιόρισμό βημάτων,



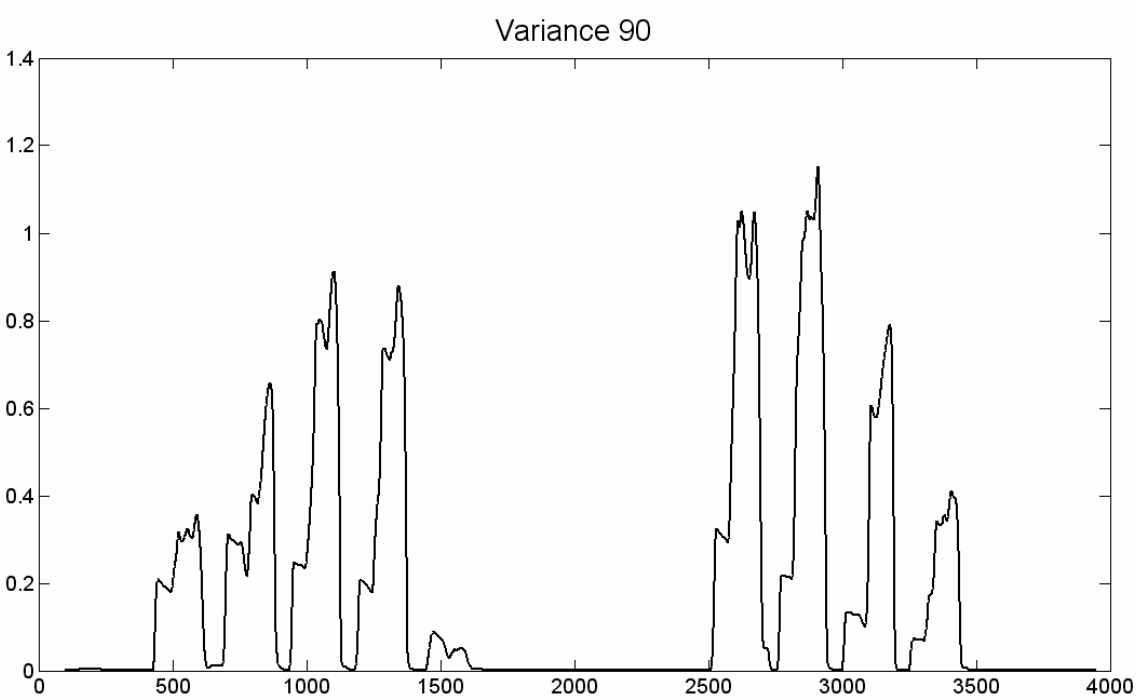
Σχήμα 10-16: Η διακύμανση με αριθμό δειγμάτων 60 αρχίζει να εμφανίζει κάποια ευκολία στον προσδιόρισμό βημάτων,



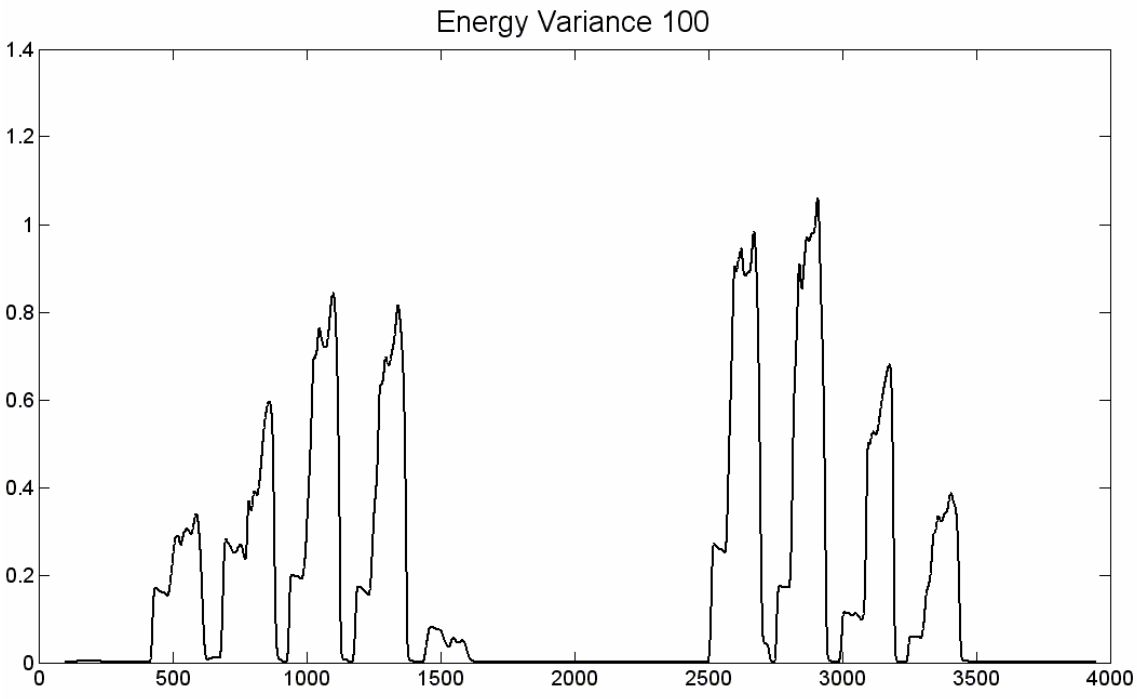
Σχήμα 10-17: Η διακύμανση με αριθμό δειγμάτων 70 μπορεί να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.



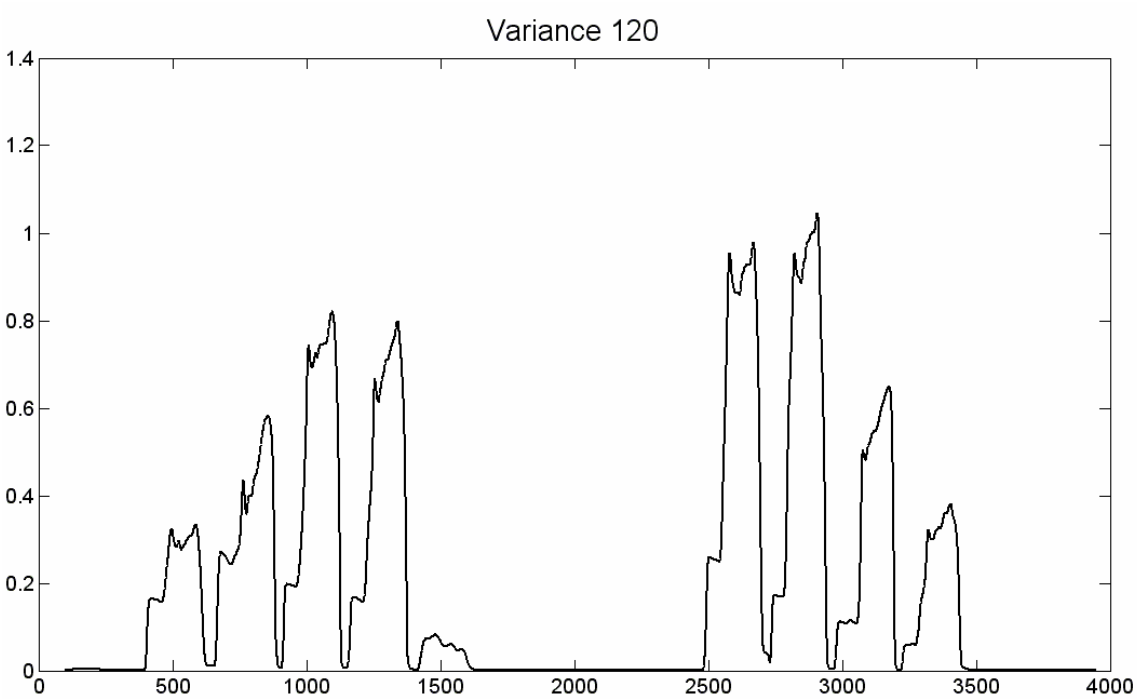
Σχήμα 10-18: Η διακύμανση με αριθμό δειγμάτων 80 μπορεί να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.



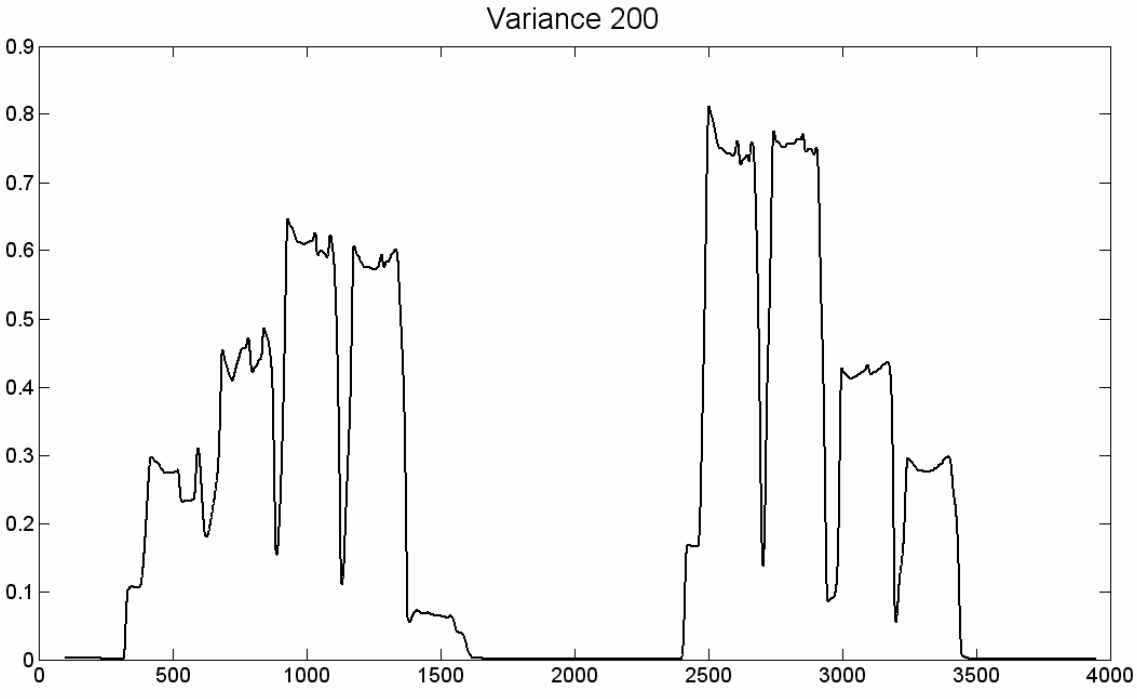
Σχήμα 10-19: Η διακύμανση με αριθμό δειγμάτων 90 μπορεί να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.



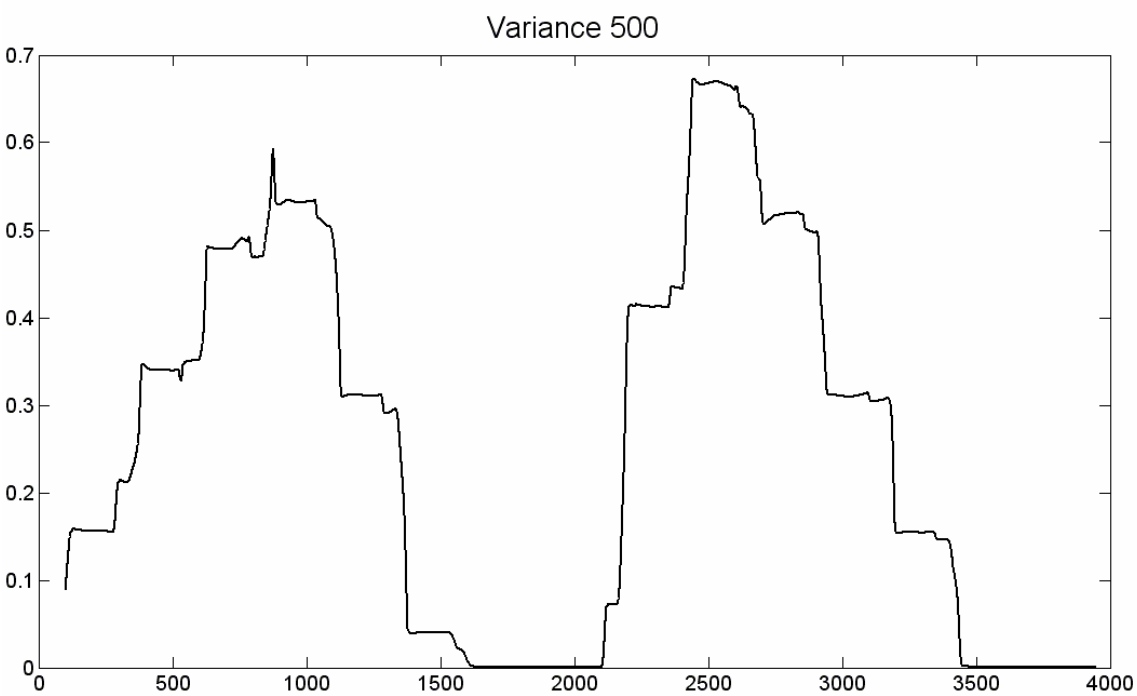
Σχήμα 10-20: Η διακύμανση με αριθμό δειγμάτων 100 μπορεί να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.



Σχήμα 10-21: Η διακύμανση με αριθμό δειγμάτων 120 μπορεί να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.



Σχήμα 10-22: Η διακύμανση με αριθμό δειγμάτων 200 μπορεί οριακά να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.

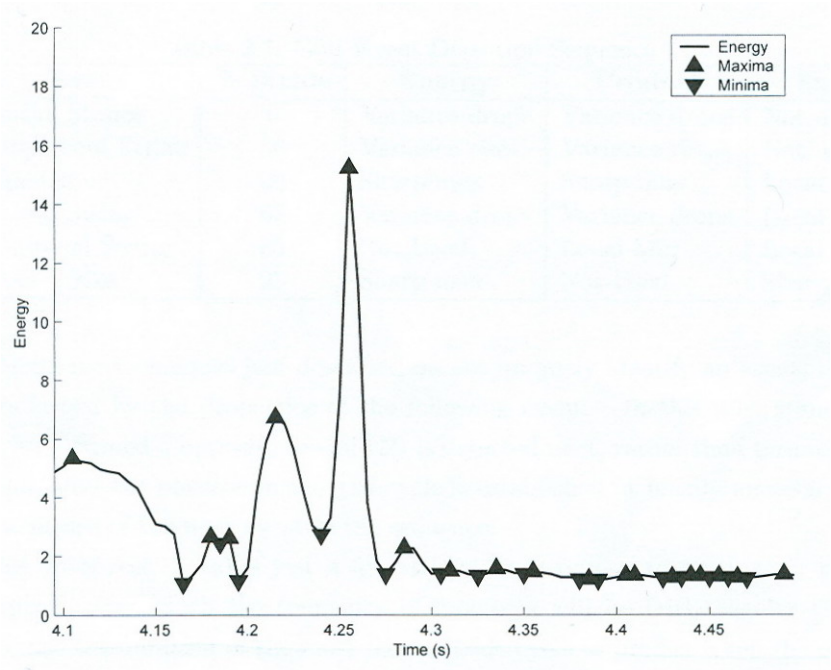


Σχήμα 10-23: Η διακύμανση με αριθμό δειγμάτων 500 δεν μπορεί να χρησιμοποιηθεί για τον διαχωρισμό βημάτων.

10.6 Τοπικά ακρότατα

Μια διαφορετική μέθοδος από τη μέθοδο της διακύμανσης για την ανίχνευση γεγονότων βηματισμού σε ένα σήμα είναι η ανίχνευση για μέγιστα και ελάχιστα. Ένας απλός ορισμός είναι: ένα σημείο είναι μέγιστο όταν έχει την μεγαλύτερη τιμή από τα σημεία που υπάρχουν πριν και μετά από αυτό. Για το ελάχιστο πρέπει το σημείο να έχει την μικρότερη τιμή από τα υπόλοιπα. Αυτό υλοποιείται πολύ εύκολα προγραμματιστικά καθώς μπορούμε κάθε σημείο να το συγκρίνουμε με το προηγούμενο και το επόμενό του. Το σχήμα 10-25 δείχνει ένα μικρό μέρος του σήματος ενέργειας. Τα τριγωνικά σηματα που δείχνουν προς τα πάνω σημαδεύουν τα τοπικά μέγιστα ενώ αυτά που δείχνουν προς τα κάτω δείχνουν τα τοπικά ελάχιστα. Αρχικά μπορεί να φανεί ότι χρήσιμες κορυφές όπως αυτή του σημείου t=4,25s δυσκολεύονται να απομονωθούν λόγω των διαταραχών ακροτάτων λόγω θορύβου μιας σταθερής περιόδου όπως αυτής μετά τα t=4,3s. Ωστόσο, αυτό σημαίνει ότι η συχνότητα και η σχετική τιμή των ακροτάτων που βρίσκονται με αυτή τη μέθοδο μπορούν να χρησιμοποιηθούν για την εύρεση διαφόρων γεγονότων βηματισμού.

Όταν το σήμα δεν αλλάζει, τα τοπικά μέγιστα και ελάχιστα θα έχουν περίπου το ίδιο πλάτος και θα συμβαίνουν σε γρήγορες εναλλαγές εξαιτίας του θορύβου του σήματος. Ένα αληθινό μέγιστο θα έχει πολύ μεγαλύτερη τιμή πλάτους από τα ελάχιστα που το περιβάλλουν. Το ακρότατο που υπάρχει μετά από μια περίοδο μονοτονικών αλλαγών, θα ξεχωρίσει από τα προηγούμενα ελάχιστα καθώς θα διαρκέσει περισσότερο.



Σχήμα 10-24: Εύρεση τοπικών μεγίστων και ελαχίστων σε ένα σήμα.

10.7 Ακολουθία γεγονότων βηματισμού

Ένας άνθρωπος που περπατάει κανονικά, ακολουθεί ένα συγκεκριμένο μοτίβο κίνησης, που σημαίνει ότι τα γεγονότα βηματισμού λαμβάνουν χώρα σε καθορισμένη σειρά. Μόλις η ρουτίνα ανίχνευσης βηματισμού ανιχνεύσει σωστά ένα γεγονός βηματισμού, το επόμενο γεγονός είναι γνωστό.

Για να αναλύσουμε τον κύκλο βηματισμού, τα σήματα μετατροπής ελέγχονται για ένα συγκεκριμένο γεγονός όπως η φάση αρχικής στάσης. Αν η διακύμανση (και τα τοπικά μέγιστα και ελάχιστα) δεν αναγνωρίζουν ένα γεγονός, τότε το γεγονός μπορεί να επιβεβαιωθεί από τις ιδιότητες των ακόλουθων γεγονότων. Με αυτό τον τρόπο, η φάση αρχικής στάσης επιβεβαιώνεται αν ανιχνευτεί μετά η φάση χτυπήματος του αντίθετου ποδιού. Όταν η φάση του κύκλου βηματισμού επαληθευτεί, μόνο ο έλεγχος για τις συνθήκες του επόμενου γεγονότος είναι απαραίτητος.

Αρχίζοντας από την ηρεμία, ένας άνθρωπος χρειάζεται μόνο μερικά βήματα για να αναπτύξει την επιθυμητή ταχύτητα. Σε αυτή την ταχύτητα η συχνότητα των βημάτων του είναι αρκετά σταθερή. Είναι δυνατό να χρησιμοποιήσουμε την διάρκεια του πιο πρόσφατου κύκλου βηματισμού για να προβλέψουμε πότε θα γίνει το επόμενο χτύπημα ποδιού, για να απορρίψουμε λανθασμένες ανιχνεύσεις βηματισμών και για να υποδείξουμε πότε ένας βηματισμός είναι πιθανόν να μην ανιχνεύτηκε. Η τεχνική αυτή μπορεί να εφαρμοστεί κατά τη διάρκεια του βηματισμού για να επιταχυνθεί η ανίχνευση, μειώνοντας την περίοδο στην οποία ένα συγκεκριμένο γεγονός αναμένεται να συμβεί.

	Γεγονός	% Βηματισμού	Ενέργεια	Γινόμενο	Άθροισμα
A	Αρχική στάση	0	Δεν χρησιμοποιείται	Δεν χρησιμοποιείται	Δεν χρησιμοποιείται
B	Χτύπημα αντίθετου ποδιού	50	Αύξηση	Μείωση	Δεν χρησιμοποιείται
C	Ανύψωση ποδιού	60	Απότομο μέγιστο	Απότομο ελάχιστο	Τοπικό μέγιστο
D	Αρχική αιώρηση	65	Ελάχιστο	Απότομο μέγιστο	Δεν χρησιμοποιείται
F	Τελική αιώρηση	85	Μείωση	Τοπικό μέγιστο	Μέγιστο
H	Χτύπημα ποδιού	95	Απότομο μέγιστο	Τοπικό ελάχιστο	Απότομο μέγιστο

Πίνακας 10-a: Γεγονότα βηματισμού.

10.8 Διαδικασία ανίχνευσης γεγονότων βηματισμού

Οι τεχνικές των σημάτων μετατροπής, όπως η διακύμανση, τα τοπικά μέγιστα και ελάχιστα και η ακολουθία γεγονότων, μπορούν να χρησιμοποιηθούν για την πιο επιτυχημένη ανίχνευση των γεγονότων βηματισμού. Ο πίνακας 10-a περιγράφει τα γεγονότα βηματισμού με την σειρά που συμβαίνουν και προσεγγιστικά την περίοδο που αναμένεται να συμβούν. Επίσης εμφανίζονται οι καταστάσεις των πλατών και των τοπικών μεγίστων και ελαχίστων στα σήματα ενέργειας, γινομένου και αθροίσματος. Όπου είναι δυνατό, η επιτυχία της ανίχνευσης αυξάνεται ελέγχοντας πολλαπλά σήματα και μετά χρησιμοποιώντας τον μέσο χρόνο ανίχνευσης.

Η ανίχνευση της πρώτης φάσης στάσης είναι το πιο σημαντικό βήμα στην όλη διαδικασία καθώς έπειτα απλά ψάχνουμε ένα μικρό παράθυρο δειγμάτων για τις συνθήκες του επόμενου γεγονότος βηματισμού. Μετά την πρώτη ανίχνευση, το σήμα που έχει προηγηθεί πριν από την φάση στάσης μπορεί να ελεγχθεί αντίστροφα για γεγονότα βηματισμού μέχρι να φτάσουμε στην αρχή των δειγμάτων που πήραμε με δειγματοληψία.

Μια από τις διαδικασίες που μπορούν να χρησιμοποιηθούν είναι η εξής. Μετά από τη συλλογή ενός ικανοποιητικού αριθμού δειγμάτων δημιουργούμε τα σήματα που έχουμε περιγράψει και υπολογίζουμε την διακύμανση για n=100. Εξετάζουμε πόσες φορές η τιμή της διακύμανσης ανεβαίνει πάνω από το κατώφλι που έχουμε επιλέξει και με αυτό τον τρόπο υπολογίζουμε τον αριθμό των κύκλων βηματισμού που έχουν γίνει.

Αν θέλουμε να εντοπίσουμε τα διάφορα γεγονότα βηματισμού σε κάθε βήμα απόμονώνουμε έναν κύκλο βηματισμού κάθε φορά. Κάθε κύκλος είναι και ένας “παλμός” όπως φαίνεται από το σχήμα 10-20.

10.8.1 Ανίχνευση φάσης τελικής αιώρησης

Από το σήμα αθροίσματος επιτάχυνσης βρίσκουμε το μέγιστο το οποίο αντιστοιχεί στο σημείο της τελικής αιώρησης.

10.8.2 Ανίχνευση φάσης αρχής στάσης

Στο σήμα γινομένου ψάχνουμε μετά το γεγονός τελικής αιώρησης για το σημείο που το σήμα αρχίζει να έχει για μεγάλο διάστημα μηδενική τιμή.

10.8.3 Ανίχνευση χτυπήματος ποδιού στο έδαφος

Στο σήμα γινομένου ψάχνουμε μεταξύ του σημείου τελικής αιώρησης και αρχής στάσης για το ελάχιστο. Αυτό αντιστοιχεί στο χτύπημα του δεξιού ποδιού στο έδαφος.

10.8.4 Ανίχνευση φάσης αρχικής αιώρησης

Στο σήμα Υ επιτάχυνσης ψάχνουμε πριν το σημείο τελικής αιώρησης για το ελάχιστο σημείο. Αυτό αντιστοιχεί στη φάση αρχικής αιώρησης και είναι εύκολο να ανιχνευτεί γιατί βρίσκεται μεταξύ δύο κορυφών.

10.8.5 Ανίχνευση φάσης ανύψωσης ποδιού

Στο σήμα Υ επιτάχυνσης ψάχνουμε πριν από τη φάση αρχικής αιώρησης. Το μέγιστο που θα βρούμε αντιστοιχεί στην ανύψωση του ποδιού από το έδαφος.

10.8.6 Ανίχνευση χτυπήματος αντίθετου ποδιού στο έδαφος

Ψάχνουμε στο σήμα ενέργειας πριν από τη φάση ανύψωσης του ποδιού από το έδαφος για το σημείο που το σήμα μηδενίζεται. Αυτό το σημείο αντιστοιχεί στο χτύπημα του αντίθετου ποδιού στο έδαφος.

Αυτή είναι μια πολύ απλή αλλά αξιόπιστη μέθοδος ανίχνευσης των γεγονότων βηματισμού που βασίζεται αρχικά στην ανίχνευση του τελικού γεγονότων και στην συνέχεια βρίσκοντας μέγιστα ελάχιστα και σχετικές τιμές μεταξύ των γεγονότων μπορεί και ανιχνεύει όλα τα γεγονότα. Μπορούν να δημιουργηθούν και άλλες μέθοδοι που βασίζονται στα σήματα που έχουμε δημιουργήσει και στην ανάλυση που έχουμε κάνει προηγουμένως.

10.9 Συγκριση δεδομένων

Για να μελετήσουμε τα σήματα επιταχύνσεων πήραμε μετρήσεις από δύο άντρες 23 ετών και από μια γυναίκα 24 ετών. Τα αποτελέσματα των μετρήσεων και τα διάφορα σήματα παρουσιάζονται παρακάτω. Παρατηρούμε ότι τα σήματα έχουν πολλές ομοιότητες και σε γενικές γραμμές έχουν την ίδια μορφή κατά τη διάρκεια των διαφόρων γεγονότων βηματισμού.

	Ύψος	Βάρος
Άνδρας Α	1.90m	75kg
Άνδρας Β	1.82m	118kg
Γυναίκα	1.67m	65kg

Πίνακας 10-b: Βασικά χαρακτηριστικά ανθρώπων που πήραν μέρος στις μετρήσεις.

10.9.1 Σύγκριση σημάτων Y επιτάχυνσης

Η επιτάχυνση Y ξεκινάει με το σήμα σε σχετικά σταθερό πλάτος. Μόλις συμβεί το γεγονός του χτυπήματος του αντίθετου ποδιού υπάρχει μια μικρή ανύψωση. Στην συνέχεια ακολουθεί η ανύψωση του δεξιού ποδιού που έχουμε τον αισθητήρα. Τότε παρατηρείται μια μεγάλη ανύψωση στο σήμα, διότι το πόδι δίνει ώθηση για να φύγει μπροστά, στην οποία η κορυφή της σηματοδοτεί την ανύψωση του ποδιού από το έδαφος και έπειτα έχουμε μια απότομη μείωση στην επιτάχυνση που το ελάχιστο είναι η αρχή της φάσης αιώρησης. Κατά τη διάρκεια της φάσης αιώρησης η επιτάχυνση αυξάνεται συνεχώς, στην αρχή πιο γρήγορα και κατά το τέλος πιο αργά διότι το πόδι φτάνει στον προορισμό του. Το τέλος της φάσης αιώρησης είναι και το μέγιστο αυτής της ανύψωσης. Αμέσως μετά έχουμε ένα απότομα βύθισμα στην οποία η επιτάχυνση πάει στο ονομαστικό της επίπεδο. Αυτό συμβαίνει διότι το πόδι έχει φτάσει πάνω από το σημείο που θα χτυπήσει στο έδαφος και συνεπώς δεν υπάρχει επιτάχυνση στον Y άξονα. Μόλις χτυπήσει το πόδι στο έδαφος έχουμε απότομη ανύψωση και στην συνέχεια η επιτάχυνση πέφτει στο ονομαστικό της επίπεδο που συμβαίνει επειδή το πόδι έχει σταθεροποιηθεί στο έδαφος.

Οι διαφορές στην Y επιτάχυνση βρίσκονται κυριώς στα διαφορετικά πλάτη και στις διάρκειες στα διάφορα γεγονότα βηματισμού. Για παράδειγμα η ανύψωση μετά το χτύπημα του αριστερού ποδιού είναι πιο εμφανής στον άνδρα Α. Επίσης στο χτύπημα του ποδιού η επιτάχυνση είναι πιο μεγάλη στους άντρες λόγω του πιο δυνατού βηματισμού τους. Στο σήμα της γυναίκας μπορούμε να παρατηρήσουμε ότι είναι πιο σταθερό γιατί δεν υπάρχουν οι μικρές διακυμάνσεις πάνω στο σήμα.

10.9.2 Σύγκριση σημάτων Z επιτάχυνσης

Η επιτάχυνση Z ξεκινάει με το σήμα στα ονομαστικά επίπεδα. Μετά το χτύπημα του αριστερού ποδιού αρχίζει μια ελαφριά μείωση στην επιτάχυνση διότι το δεξί πόδι πιέζεται προς τα κάτω πριν την ανύψωση. Η ανύψωση

ξεκινάει όταν η επιτάχυνση έχει την ελάχιστη τιμή. Στην φάση της αιώρησης η επιτάχυνση στην αρχή αυξάνεται, μετά μειώνεται και τέλος αυξάνεται πάλι. Αυτό συμβαίνει λόγω της καμπύλης που διαγράφει το πόδι στον αέρα. Όταν αρχίσει το πόδι να πέφτει για να χτυπήσει στο έδαφος, έχουμε απότομη μείωση της επιτάχυνσης. Κατά το χτύπημα παρατηρείται μια αιχμή και στην συνέχεια μια μικρή ταλάντωση που οφείλεται στην απορρόφηση του κραδασμού από το πόδι και το παπούτσι. Μετά από την ταλάντωση, η επιτάχυνση γυρίζει στα ονομαστικά της επίπεδα διότι έχουμε ηρεμία στο πόδι.

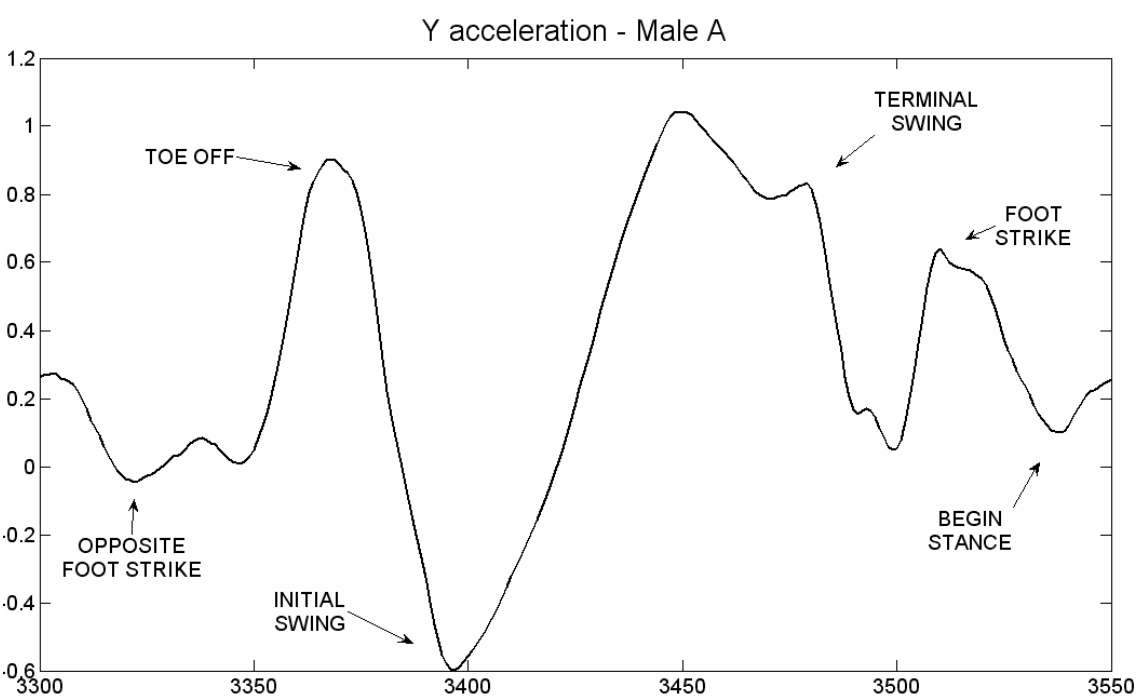
Οι διαφορές στα σήματα της επιτάχυνσης Z εμφανίζονται κυριώς στην φάση της αιώρησης καθώς στο σήμα του άνδρα Α εμφανίζεται πιο έντονα η καμπύλη της αιώρησης. Αυτό μπορεί να οφείλεται στο ότι είχε μεγαλύτερα πόδια από τους υπόλοιπους ανθρώπους με αποτέλσμα να διαγράφει μεγαλύτερη καμπύλη το πόδι του στον αέρα με μεγαλύτερη ροπή. Ομοίως το χτύπημά του στο έδαφος είναι πιο δυνατό και αργεί πιο πολύ να γίνει η απόσβεσή του, εμφανίζοντας καθαρά την ταλάντωση. Ο άνδρας Β χτυπάει το πόδι στο έδαφος με μικρότερη δύναμη γι' αυτό η ταλάντωση διαρκεί λιγότερο από ότι στον Α. Στο σήμα της γυναίκας, επειδή έχει πιο αδύναμο βηματισμό, το χτύπημα δεν είναι δυνατό και η απόσβεση του χτυπήματος γίνεται σχεδόν αμέσως.

10.9.3 Σύγκριση σημάτων ενέργειας επιτάχυνσης

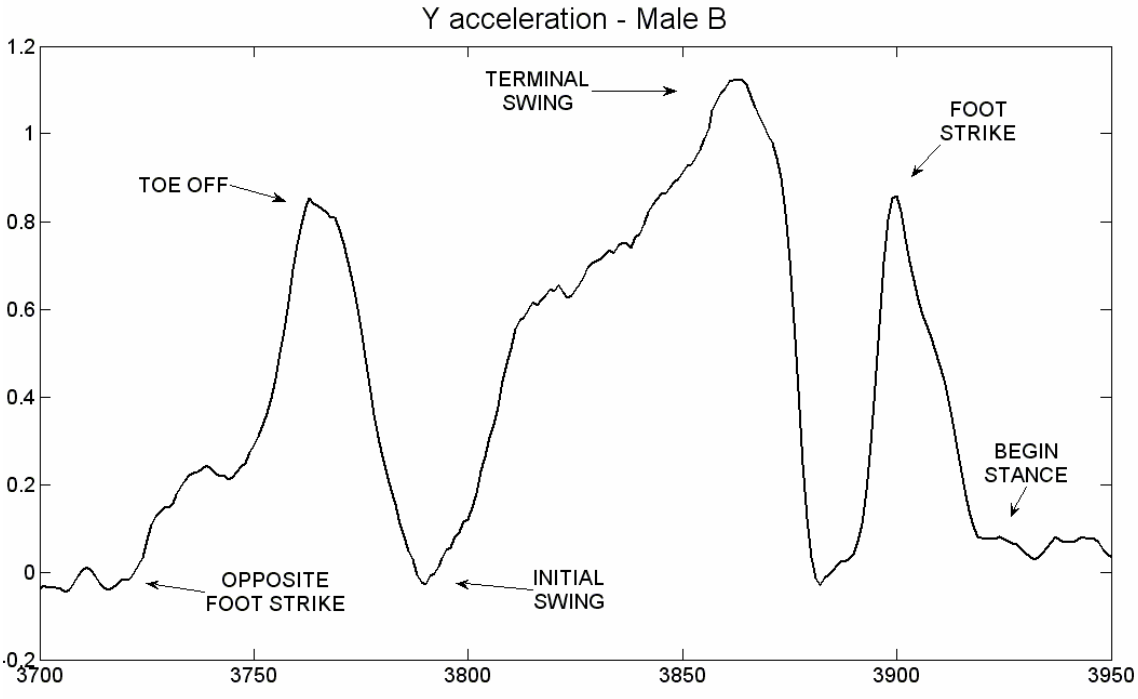
Το σήμα ενέργειας επιτάχυνσης παρουσιάζει μια μικρή ανύψωση μολίς γίνει το χτύπημα του αριστερού ποδιού. Στην συνέχεια η ενέργεια αυξάνεται απότομα διότι το πόδι πιέζει το έδαφος και πάει να ανυψωθεί. Στο μέγιστο αυτής της ενέργειας το πόδι φεύγει από το έδαφος. Η ενέργεια πέφτει εξίσου απότομα όταν το πόδι βρεθεί στον αέρα. Τότε ξεκινάει η περίοδος αιώρησης με την ενέργεια στο ελάχιστο. Αρχικά καθώς γίνεται η αιώρηση προς τα εμπρός και προς τα πάνω του ποδιού, η ενέργεια αυξάνεται γρήγορα ενώ όταν το πόδι φτάνει στο μέγιστο ύψος ο άνθρωπος μειώνει την δύναμη που ασκεί με αποτέλεσμα να μειώνεται και η ενέργεια επιτάχυνσης ενώ όταν το πόδι αρχίσει να πέφτει προς το έδαφος η ενέργεια μειώνεται πολύ γρήγορα. Κατά το χτύπημα του δεξιού ποδιού στο έδαφος έχουμε μια πολύ γρήγορη έως απότομη αύξηση του σήματος. Στην συνέχεια το σήμα μειώνεται πιο αργά μέχρι να φτάσει την τιμή ηρεμίας καθώς το πόδι έχει σταματήσει.

Οι διαφορές είναι ότι το σήμα της γυναίκας δεν εμφανίζει μεγάλη ανύψωση κατά το χτύπημα του δεξιού ποδιού στο έδαφος γιατί το χτυπάει με λιγότερη δύναμη. Επίσης αξίζει να αναφέρουμε ότι η γυναίκα ζύγιζε λιγότερο από τους άνδρες και αυτό επηρεάζει το χτύπημα στο έδαφος.

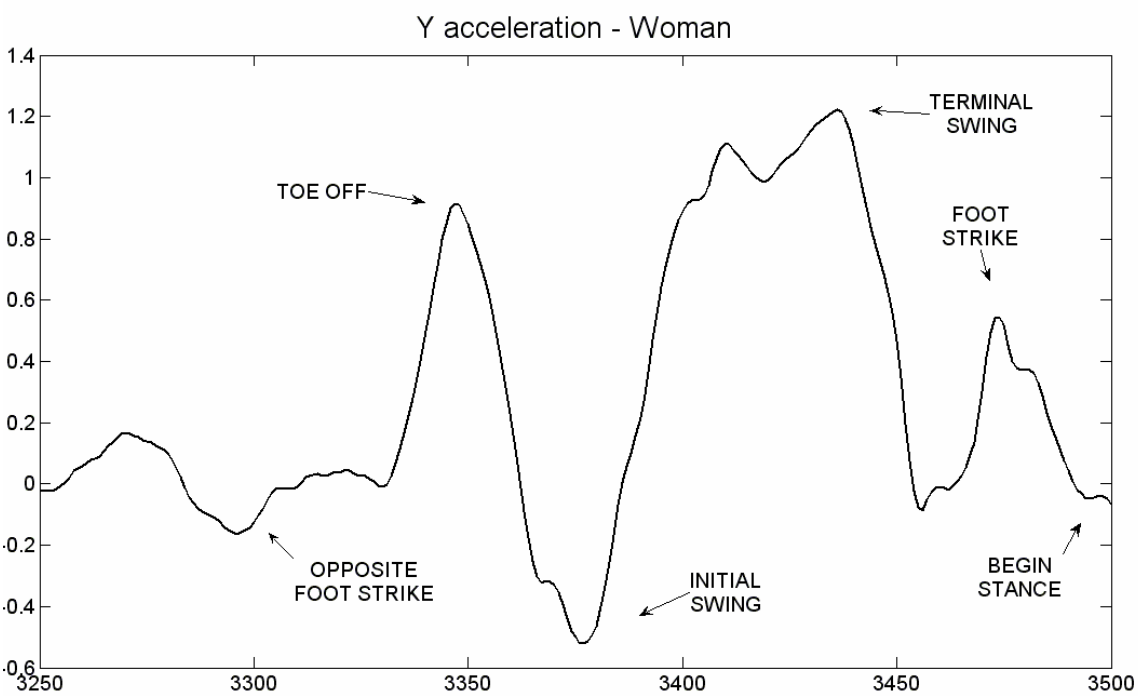
Από τα υπόλοιπα σήματα πάλι μπορούμε να δούμε τις κύριες ομοιότητες και μικρές διαφορές καθώς επίσης και τα χαρακτηριστικά γεγονότα που φαίνονται πιο καθαρά με τα μέγιστα και τα ελάχιστα.



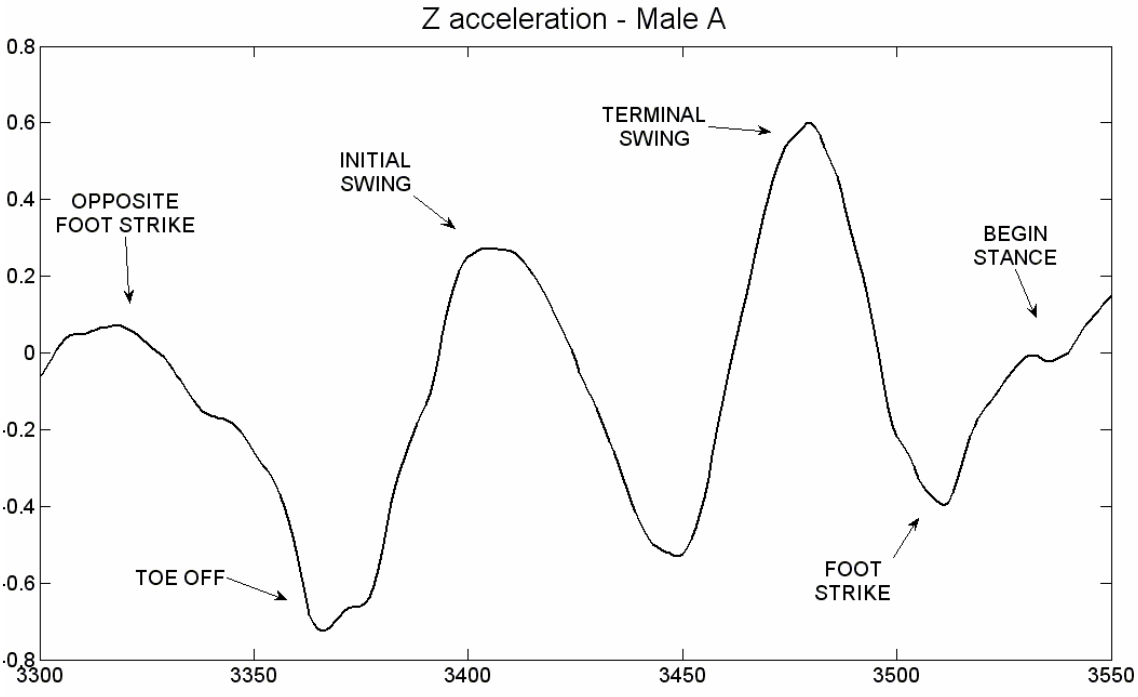
Σχήμα 10-25: Επιτάχυνση Y του άντρα Α.



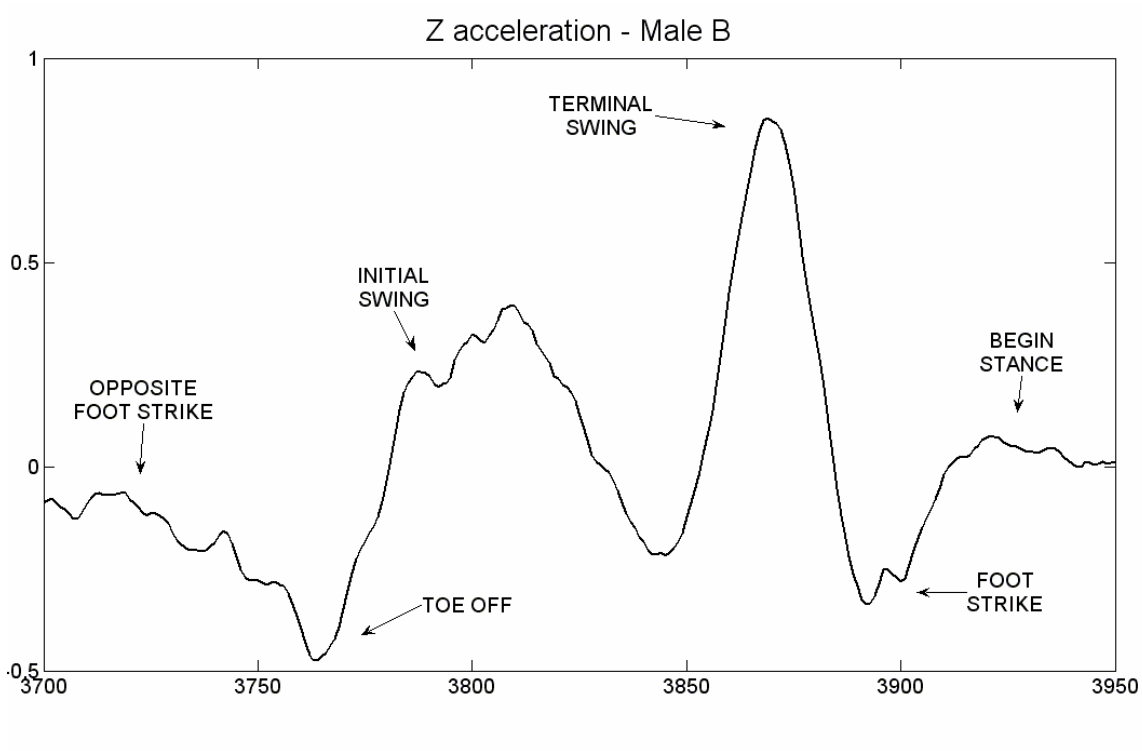
Σχήμα 10-26: Επιτάχυνση Y του άντρα Β.



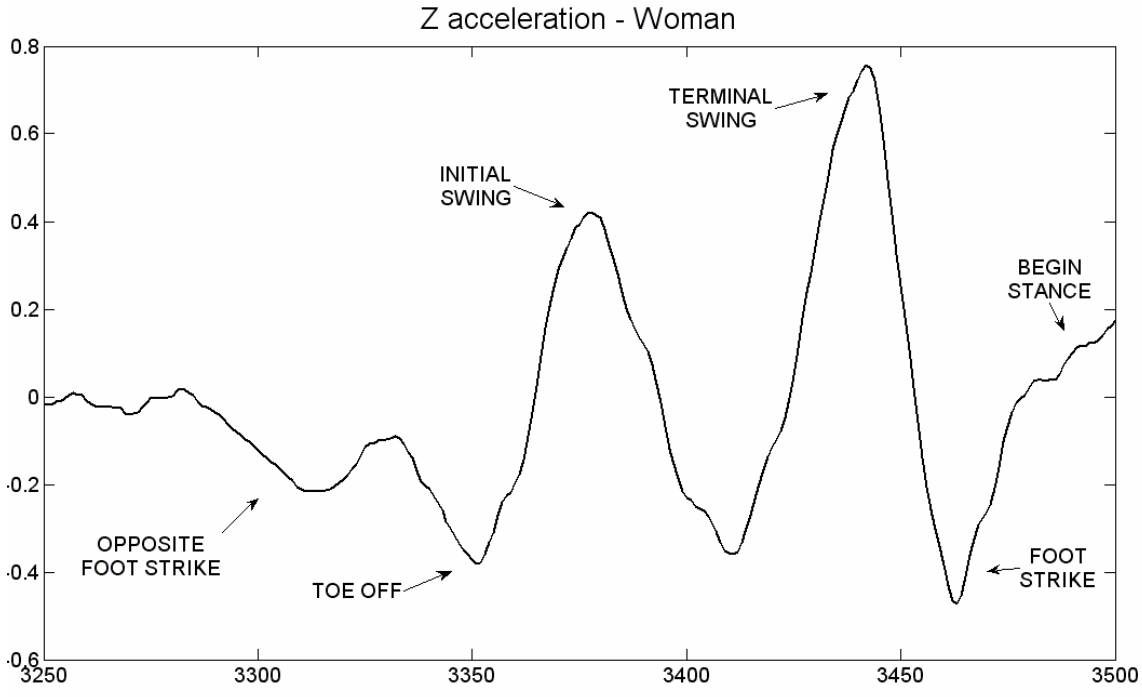
Σχήμα 10-27: Επιτάχυνση Y της γυναίκας.



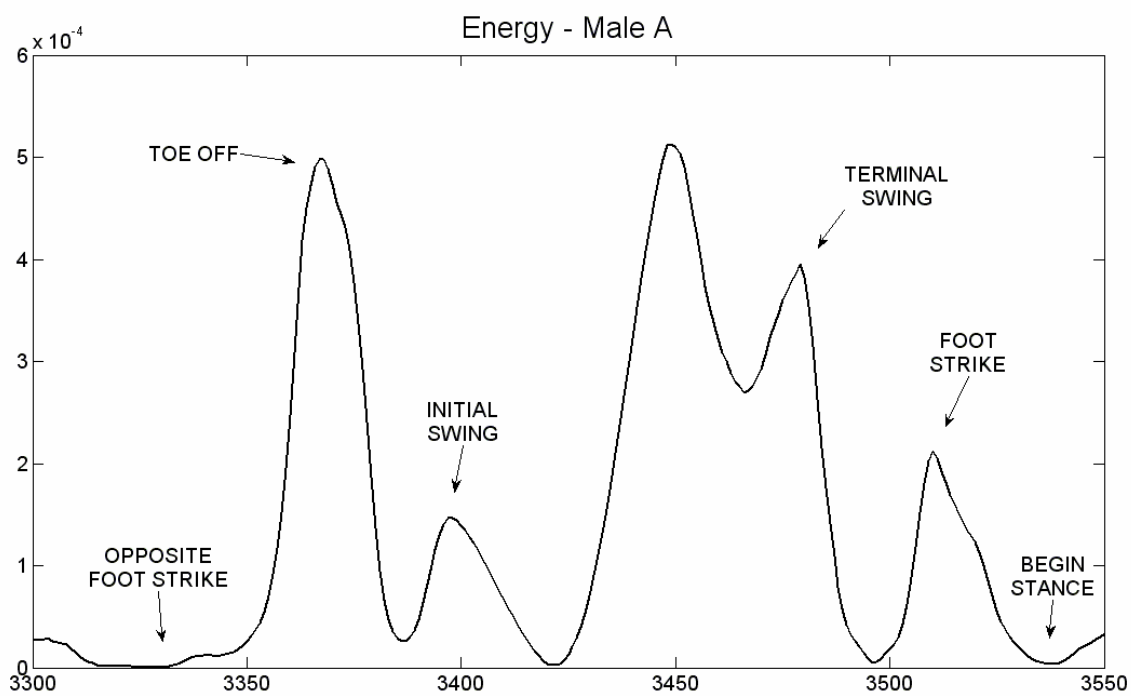
Σχήμα 10-28: Επιτάχυνση Z ανθρώπου Α.



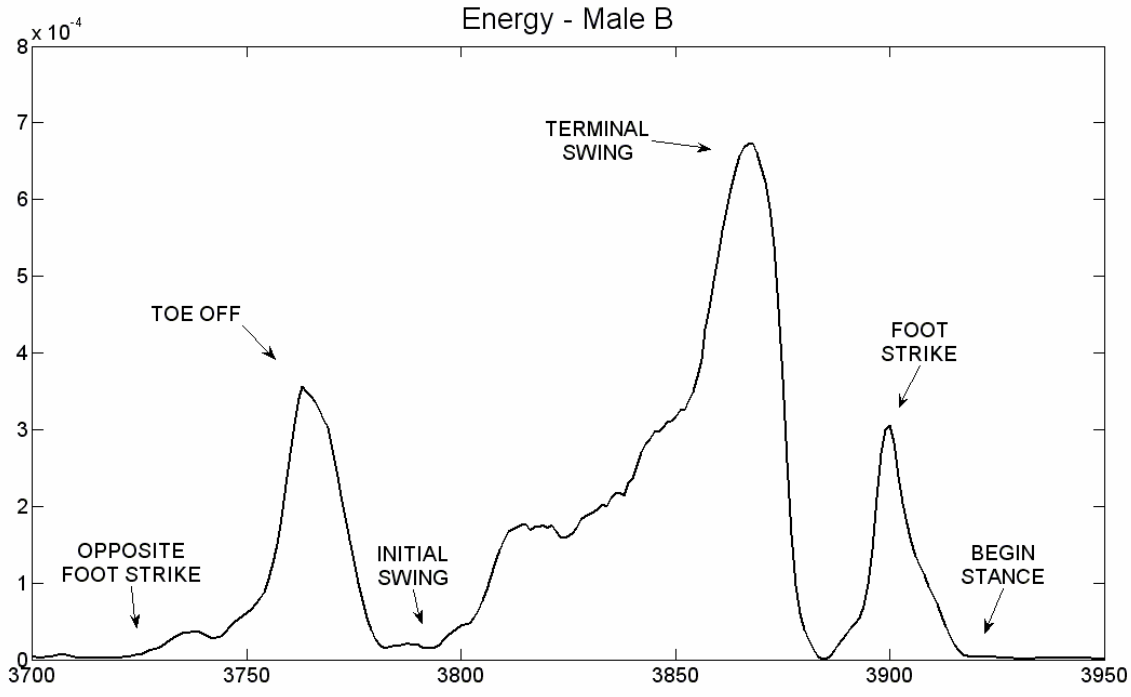
Σχήμα 10-29: Επιτάχυνση Z του ανθρώπου Β.



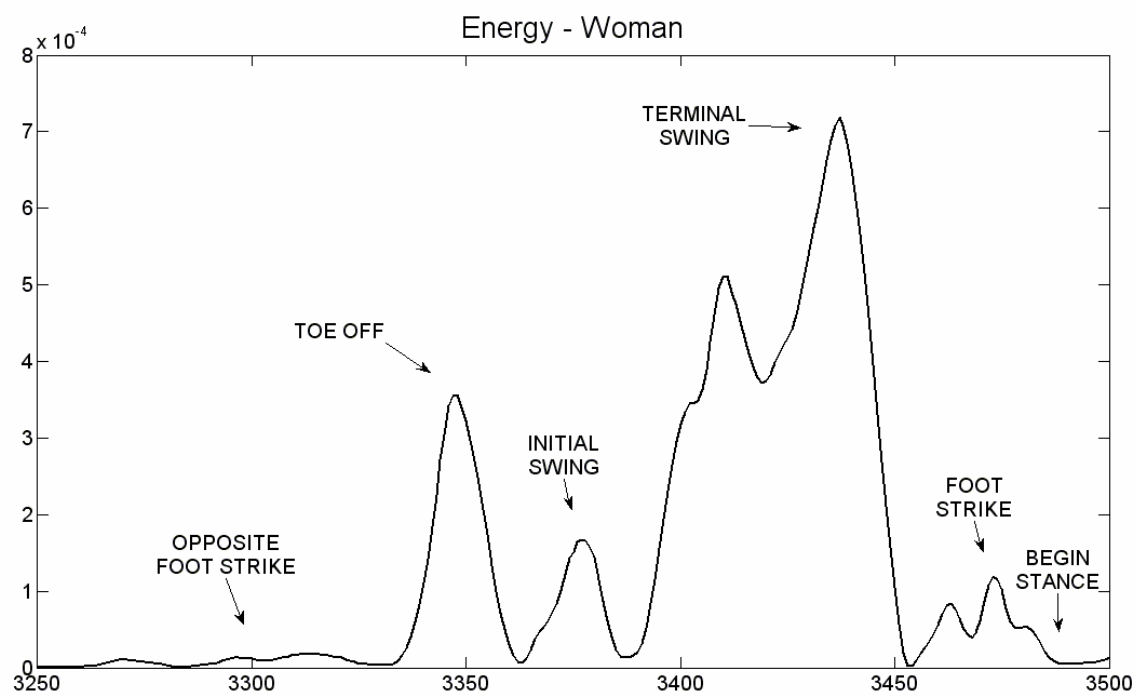
Σχήμα 10-30: Επιτάχυνση Z της γυναίκας.



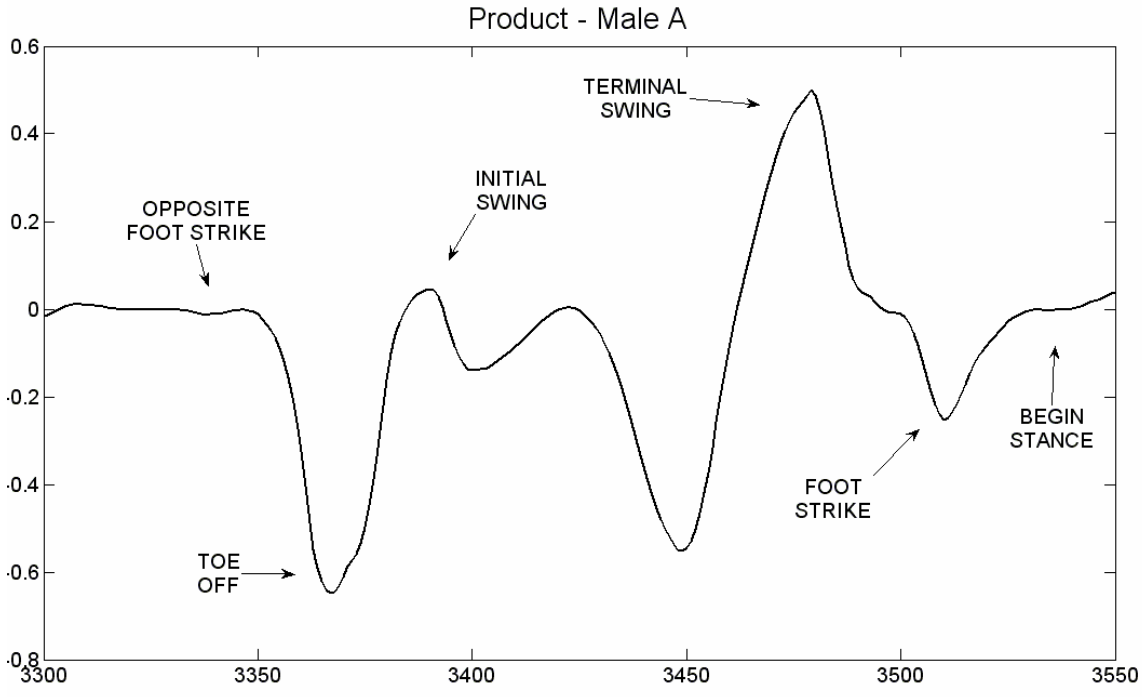
Σχήμα 10-31: Ενέργεια επιτάχυνσης ανθρώπου Α.



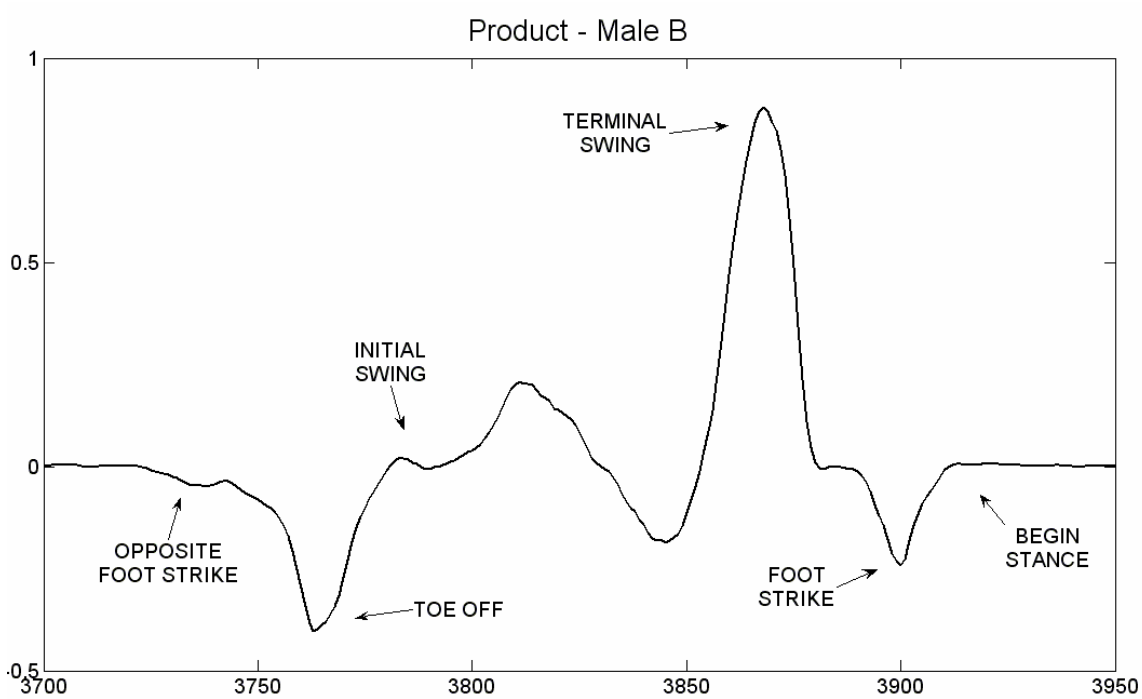
Σχήμα 10-32: Ενέργεια επιτάχυνσης ανθρώπου Β.



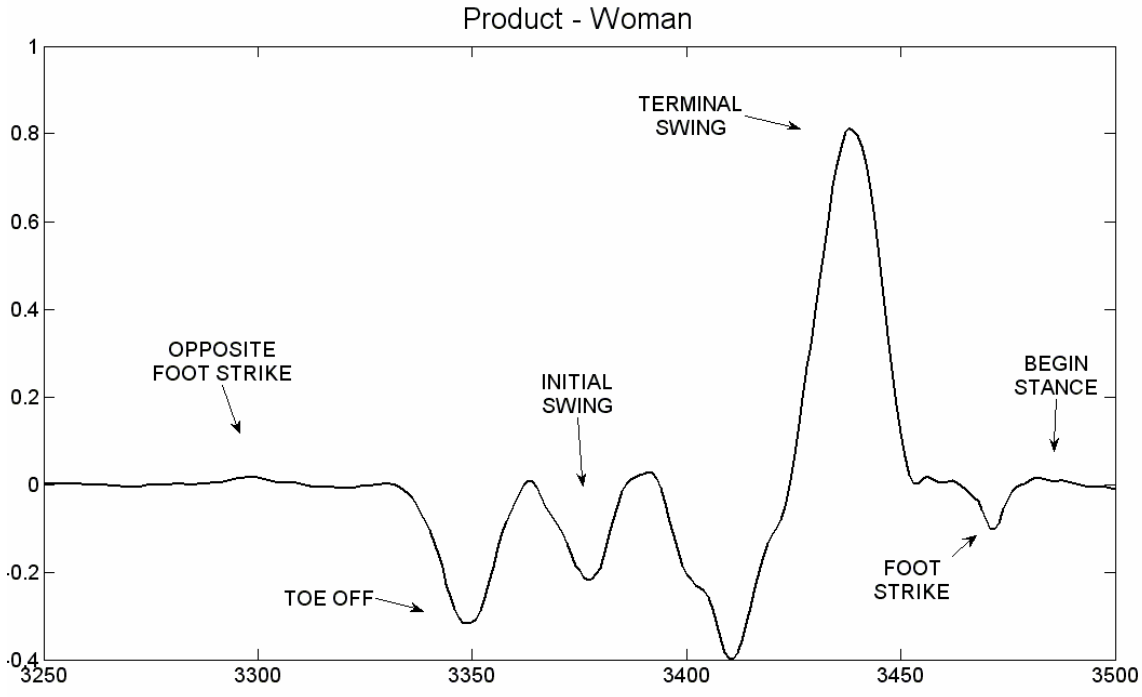
Σχήμα 10-33: Ενέργεια επιτάχυνσης της γυναίκας.



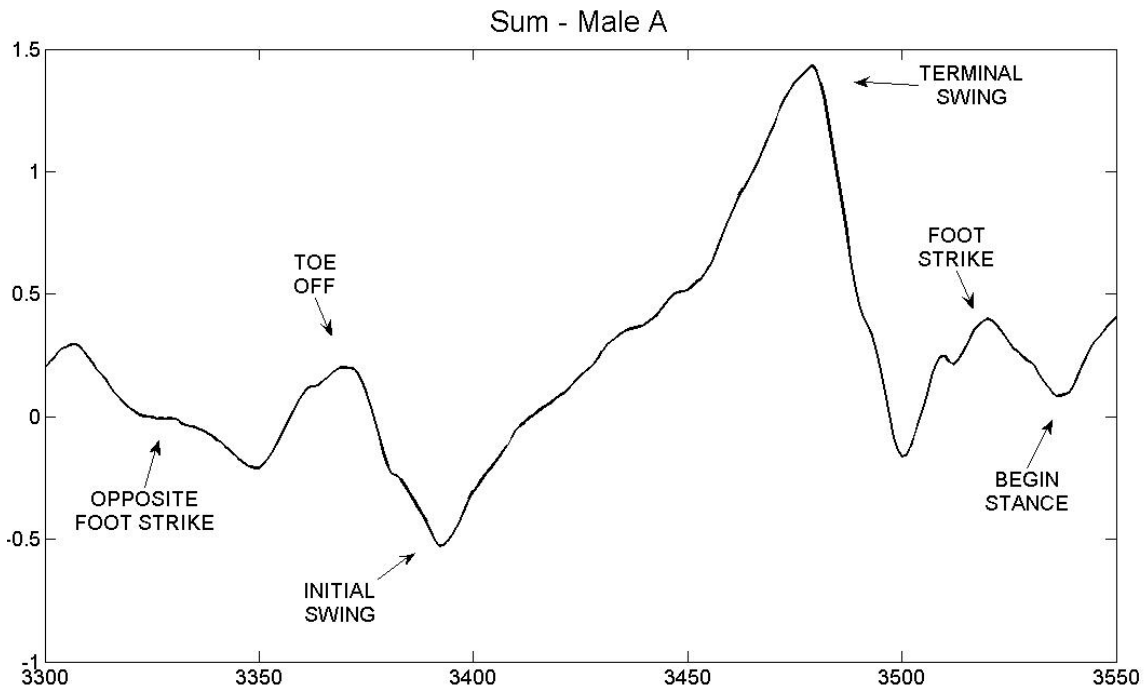
Σχήμα 10-34: Γινόμενο επιτάχυνσης ανθρώπου Α.



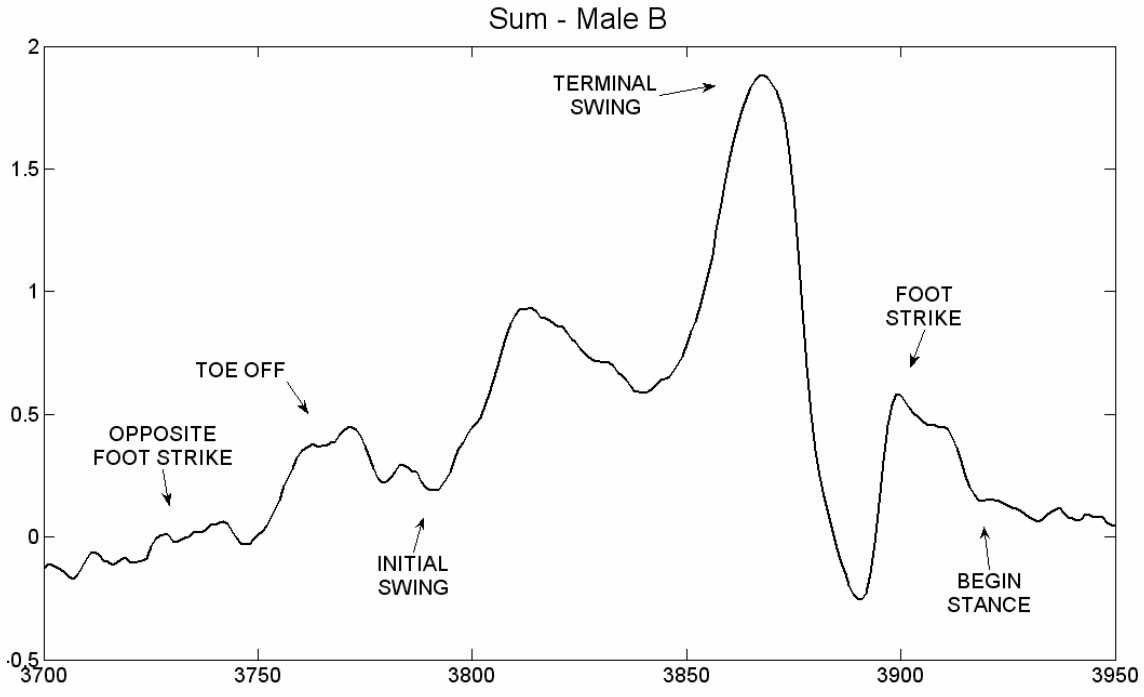
Σχήμα 10-35: Γινόμενο επιτάχυνσης ανθρώπου Β.



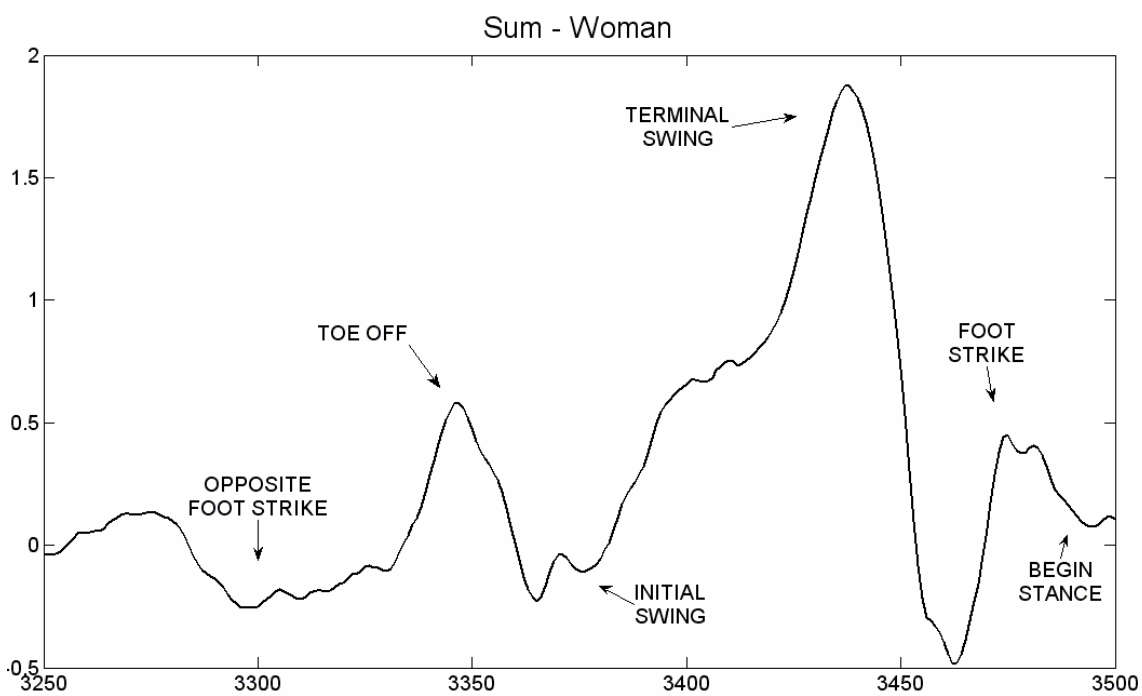
Σχήμα 10-36: Γινόμενο επιτάχυνσης της γυναίκας.



Σχήμα 10-37: Άθροισμα επιτάχυνσης ανθρώπου Α.



Σχήμα 10-38: Άθροισμα επιτάχυνσης ανθρώπου Β.



Σχήμα 10-39: Άθροισμα επιτάχυνσης της γυναίκας.

10.10 Συνοπτικά

Ένας αισθητήρας επιτάχυνσης τοποθετημένος στο πόδι μπορεί να χρησιμοποιηθεί για να ανιχνεύσει συγκεκριμένα γεγονότα του ανθρώπινου κύκλου βηματισμού. Επειδή η ακολουθία αυτών των γεγονότων είναι σταθερή μεταξύ των βηματισμών, το σήμα επιτάχυνσης μπορεί να ελεγθεί για χαρακτηριστικά μοτίβα ώστε να διακρίνουμε την πραγματοποίηση γεγονότων βηματισμού όπως το χτύπημα του ποδιού στο έδαφος, τη φάση στάσης και την ανύψωση του ποδιού. Η μετρούμενη επιτάχυνση χρησιμοποιείται κατάλληλα για να δημιουργηθούν τα σήματα ενέργειας, γινομένου και αθροίσματος που αναδεικνύουν διάφορα γεγονότα στον βηματισμό, κάνοντάς τα πιο εύκολα να αναγνωριστούν. Εφαρμόζοντας τεχνικές που είναι ευαίσθητες σε μοτίβα αλλαγών των σημάτων και όχι απλά στο πλάτος τους, ο αλγόριθμος ανίχνευσης βηματισμού μπορεί να γίνει πολύ αξιόπιστος και επιτυχής σε σχέση με τα λάθη στην βαθμονόμηση του αισθητήρα και σε αλλαγές των συνθηκών βηματισμού.

ΚΕΦΑΛΑΙΟ 11

Εφαρμογές

11.1 Εισαγωγή

Το σύστημα μέτρησης επιτάχυνσης σε τρεις ορθογώνιους άξονες μπορεί να χρησιμοποιηθεί σε διάφορες εφαρμογές τόσο αυτούσιο όσο και με μικρές αλλαγές.

11.2 Ιατρικές εφαρμογές

Μια χρήσιμη εφαρμογή είναι η παρακολούθηση της κίνησης ασθενών με διάφορα κινητικά προβλήματα και νευρομυικές διαταραχές. Μπορούμε να δούμε για παράδειγμα σε ποιο τμήμα του κύκλου βηματισμού τους παρουσιάζουν διαφορές και δυσκολίες σε σχέση με τα άτομα με κανονικό βηματισμό και να τους προταθούν οι κατάλληλες θεραπείες. Αυτό είναι σημαντικό διότι η εξέταση δεν θα γίνεται με το μάτι του γιατρού, αλλά με αντικειμενικούς παράγοντες (όπως διαγράμματα). Επίσης οι διαφορές στους τρόπους βαδίσματος μπορεί να είναι μικρές ή να βρίσκονται σε αρχικό στάδιο και να μην φαίνονται με γυμνό μάτι.

Για παράδειγμα στα άτομα με πλατυποδία μπορεί να παρατηρηθεί η κίνηση και να εξακριβωθεί η απόκλιση από τον κανονικό τρόπο περπατήματος και στη συνέχεια να προταθεί η εφαρμογή κάποιου ειδικού παπουτσιού ή πάτου στο παπούτσι για να γίνει το περπάτημα πιο φυσιολογικό. Με αυτό τον τρόπο μειώνεται η καταπόνηση των ποδιών. Στην συνέχεια μπορεί να παρατηρηθεί εκ νέου η κίνηση και να διορθωθεί περαιτέρω αν χρειάζεται.

Επιπλέον μπορεί να παρατηρηθεί η αποκατάσταση ασθενών με κάποιο τραυματισμό. Με αυτό τον τρόπο οι θεράποντες γιατροί μπορούν να έχουν καλύτερο έλεγχο και επίβλεψη της προόδου των ασθενών και να κρατούν αρχείο με ποσοτικές μετρήσεις της κινήτικότητάς τους σε διάφορες χρονικές περιόδους. Έτσι μπορούν να συγκριθούν αργότερα, αντικειμενικά, διαφορετικές θεραπείες και να μπορούν οι γιατροί να εξακριβώσουν με ποιον τρόπο επιδρά μια συγκεκριμένη θεραπεία σε κάποιους τραυματισμούς.

Επειδή τα συστήματα με τα επιταχυνσιόμετρα είναι μικρά σε μέγεθος, ελαφριά, μικρής κατανάλωσης και δεν χρειάζονται ειδικό χώρο για να γίνουν μετρήσεις, μπορούν να τοποθετηθούν σε παπούτσια, ρούχα ή σε διάφορα σημεία του σώματος ενός ανθρώπου και μπορούν να καταγράφουν για μεγάλα χρονικά διαστήματα (από ώρες έως ημέρες) την κίνησή του. Συνεπώς μπορούν να καταγράφουν τις κινήσεις των ασθενών χωρίς να χρειάζεται να είναι παρών ο

γιατρός και να επεξεργάζονται στο τέλος της περιόδου είτε από τον ίδιο είτε από κάποιον υπολογιστή με έναν αλγόριθμο. Αυτό είναι ιδιαίτερα χρήσιμο για ασθενείς με την ασθένεια του Πάρκινσον ή με το σύνδρομο Τουρέτ στους οποίους θα μπορεί να παρατηρηθεί η καθημερινή τους δραστηριότητα και να ρυθμιστούν ανάλογα τα φάρμακα ή για να ενισχυθούν οι μελέτες για την ασθένεια.

Επίσης η μελέτη για την τρισδιάστατη λειτουργία τραυματισμένων γονάτων είναι πολύ σημαντική. Μετά από την εγχείρηση πρέπει να παρακολουθηθεί η δυνατότητα κάμψης του γονάτου το οποίο είναι ιδανικό να γίνει με επιταχυνσιόμετρα προσαρμοσμένα κατάλληλα στο πόδι του ασθενή. Η σωστή και έγκαιρη αποκατάσταση του γονάτου είναι μεγάλης σημασίας καθώς μειώνονται οι πιθανότητες παρουσίασης οστεοαρθρίτιδας.

Ένα από τα πιο σημαντικά προβλήματα των ανθρώπων μεγάλης ηλικίας τα ατυχήματα και οι τραυματισμοί από πτώσεις. Έχει παρατηρηθεί ότι το ένα τρίτο με το ένα δεύτερο των ηλικιωμένων έχουν ένα περιστατικό πτώσης κάθε χρόνο. Από αυτές τις πτώσεις ένα σημαντικό ποσοστό καταλλήγει σε σοβαρούς τραυματισμού ή και σε θάνατο. Είναι λοιπόν σημαντικό να αντιμετωπιστεί αυτό το φαινόμενο. Οι λόγοι που οδηγούν στις πτώσεις είναι : ελαττωματική όραση, απώλεια ελέγχου μυών, συντονισμού, μυικής δύναμης, αναγνώρισης σημάτων που στέλνουν τα πόδια και οι αρθρώσεις και ανικανότητα απόκρισης σε απότομες αλλαγές στην ισορροπία. Ωστόσο αν και οι πτώσεις στους ηλικιωμένους είναι πολύ συνηθισμένες δεν υπάρχουν αρκετές μέθοδοι ανάλυσης βηματισμού. Οι υπολογιστικές μέθοδοι είναι πιο αντικειμενικές αλλά γίνονται μόνο σε εργαστήρια και με εξειδικευμένα μηχανήματα (πχ. διάδρομοι περπατήματος). Με την χρήση επιταχυνσιομέτρων μπορεί να γίνει ανάλυση του τρόπου βηματισμού των ηλικιωμένων και να παρατηρηθούν αποκλίσεις στην κίνηση των αρθρώσεων. Μπορούν να δημιουργηθούν συστήματα βασισμένα σε επιταχυνσιόμετρα που να τοποθετούνται σε κάποιο σημείο του σώματος του ηλικιωμένου και να τον προειδοποιεί όταν υπάρχει κάποια αστάθεια στην κίνησή του.

Μια από τις πιο χρήσιμες εφαρμογές ανάλυσης βηματισμού με χρήση επιταχυνσιομέτρων είναι για την μελέτη της ισορροπίας των ανθρώπων. Κάθε άνθρωπος έχει ένα συγκεκριμένο τρόπο βηματισμού με τον οποίο νιώθει άνετα. Αυτός ο βηματισμός έχει συγκεκριμένο μήκος βήματος και ρυθμό. Η κίνηση με διαφορετικό τρόπο βηματισμού (πιο αργά ή πιο γρήγορα, με μικρότερο βήμα ή μεγαλύτερο) μειώνει την ισορροπία και αυξάνει την κατανάλωση ενέργειας. Αυτός είναι ένας από τους λόγους για τους οποίους οι ηλικιωμένοι έχουν μειωμένη ισορροπία. Νομίζουν ότι με μικρότερο και πιο αργό βηματισμό προσέχουν πιο πολύ για να μην πέσουν αλλά με αυτό τον τρόπο αποκλείνουν από τον ονομαστικό τρόπο βηματισμού τους και χάνουν πιο εύκολα την ισορροπία τους. Με τα επιταχυνσιόμετρα μπορεί να καθοριστεί η σχέση μεταξύ ισορροπίας και ενέργειας με τον τρόπο βηματισμού.

Επιπλέον με τα επιταχυνσιόμετρα μπορεί να μελετηθεί καλύτερα, πιο εύκολα και με μεγάλη ακρίβεια η καθημερινή φυσική δραστηριότητα των ατόμων και να ληφθούν στατιστικά δεδομένα, μεγάλων χρονικών περιόδων, για την σχέση μεταξύ της υγείας και της φυσικής δραστηριότητας των ατόμων κάθε ηλικίας.

Με αυτό τον τρόπο μπορούν να καθοριστούν κάποια κατώτατα όρια φυσικής άσκησης για διάφορες ομάδες πληθυσμού ώστε να επιτυγχάνεται καλύτερη ποιότητα ζωής και υγείας.

11.3 Ερευνητικοί λόγοι και εκπαίδευση

Η χρήση των επιταχυνσιομέτρων μπορεί να χρησιμοποιηθεί στην έρευνα για την αυτοματοποιημένη αναγνώριση ενεργειών του χρήστη. Για παράδειγμα από τις επιταχύνσεις μπορούμε σε πραγματικό χρόνο να αναγνωρίζουμε αν τρέχει, αν ανεβαίνει ή κατεβαίνει σκάλες, πόσα βήματα κάνει, αν είναι ακίνητος, σε ποια στάση βρίσκεται και άλλες πολλές ενέργειες.

Επιπλέον μπορεί να γίνει αναγνώριση των χρηστών που φοράνε κάποιο μηχανισμό μέτρησης επιτάχυνσης από τους διαφόρετικούς τρόπου βηματισμού που έχουν.

Η ολισθηρότητα ενός δαπέδου μπορεί να μετρηθεί εκτός από τους συμβατικούς τρόπους μέτρησης τριβής, με την χρήση ενός επιταχυνσιομέτρου προσαρμοσμένου στην φτέρνα ενός ανθρώπου. Με την χρήση κατάλληλου αλγορίθμου επεξεργασίας σήματος και την μελέτη της ολίσθησης μετά το χτύπημα της φτέρνας στο δάπεδο σε διάφορες ταχύτητες μπορεί να προκύψει η ολισθηρότητα του δαπέδου. Με αυτό τον τρόπο μπορεί το σύστημα να υπολογίζει την ολισθηρότητα και να προειδοποιεί σε πραγματικό χρόνο τον χρήστη ότι υπάρχει κίνδυνος για πτώση.

Το σύστημα επιταχυνσιομέτρων μπορεί να χρησιμοποιηθεί για εκπαιδευτικούς λόγους. Οι μαθητές μπορούν να μελετήσουν καλύτερα και να εξακριβώσουν από μόνοι τους το φαινόμενο της επιτάχυνσης πραγματοποιώντας τα κατάλληλα πειράματα. Επιπλέον ολοκληρώνοντας την επιτάχυνση μπορούν να δουν την σχέση της με την ταχύτητα και την μετατόπιση.

Ο συνδυασμός του συστήματος επιταχυνσιομέτρων με άλλους αισθητήρες που μετρούν κάποιο είδος δραστηριότητας όπως κατανάλωση οξυγόνου και καρδιακού ρυθμού μπορεί να δώσει χρήσιμες πληροφορίες για την σχέση μεταξύ τους και αυτή η σχέση να επεκταθεί για την μέτρηση της κατανάλωσης ενέργειας ενός οργανισμού σε διάφορες καταστάσεις.

11.4 Εφαρμογές στον αθλητισμό

Τα συστήματα επιταχυνσιομέτρων μπορούν να χρησιμοποιηθούν από αθλητές ώστε να έχουν δεδομένα για την απόδοσή τους. Τα δεδομένα αυτά μπορούν να χρησιμοποιηθούν από τους προπονητές τους και να συγκριθούν με τα δεδομένα από την ιδανική εκτέλεση μιας άσκησης. Με αυτό τον τρόπο μπορούν να διορθωθούν λάθη στην τεχνική ώστε να έχουν καλύτερες επιδόσεις και να προληφθούν τραυματισμοί.

Επιπλέον τα δεδομένα αυτά μπορούν να χρησιμοποιηθούν κατά την διάρκεια ενός αγώνα για να ληφθούν πληροφορίες για την απόδοση των παικτών, την ενέργεια που καταναλώνουν, την ταχύτητά τους, τον αριθμό βημάτων που

κάνουν και άλλων δεδομένων. Αυτά μπορούν να αξιοποιηθούν από τους προπονητές τους αλλά και να πληροφορήσουν τους θεατές ώστε να παρακολουθούν το παιχνίδι με μεγαλύτερη αντίληψη και ενδιαφέρον.

Οι χρήσεις και οι εφαρμογές των συστημάτων επιταχυνσιομέτρων είναι πάρα πολλές. Αυτό οφείλεται στο πολύ χρήσιμο μέγεθος που μετράνε, στο μικρό τους μέγεθος, στην μικρή κατανάλωση, στην ευκολία χρήσης και στο χαμηλό κόστος.

ΚΕΦΑΛΑΙΟ 12

Συμπεράσματα – Προτάσεις για βελτίωση

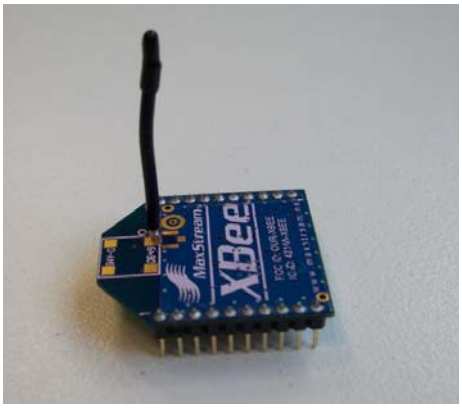
12.1 Συμπεράσματα

Η συσκευή τοποθετήθηκε στα πόδια ανθρώπων και οι μετρήσεις που πήραμε συμφωνούν με τα θεωρητικά δεδομένα. Παρατηρήθηκε ότι οι μετρήσεις χωρίς παπούτσια ήταν καλύτερες και οι διάφορες κορυφές των γεγονότων βηματισμού εμφανίζονταν πιο καθαρά. Αντιθέτως όταν οι άνθρωποι φόρούσαν παπούτσια, τα πλάτη των επιταχύνσεων ήταν μικρότερα με αποτέλεσμα να είναι πιο δύσκολη η αναγνώριση των γεγονότων. Αυτό οφείλεται στην απόσβεση που προσδίδουν οι σόλες των παπουτσιών.

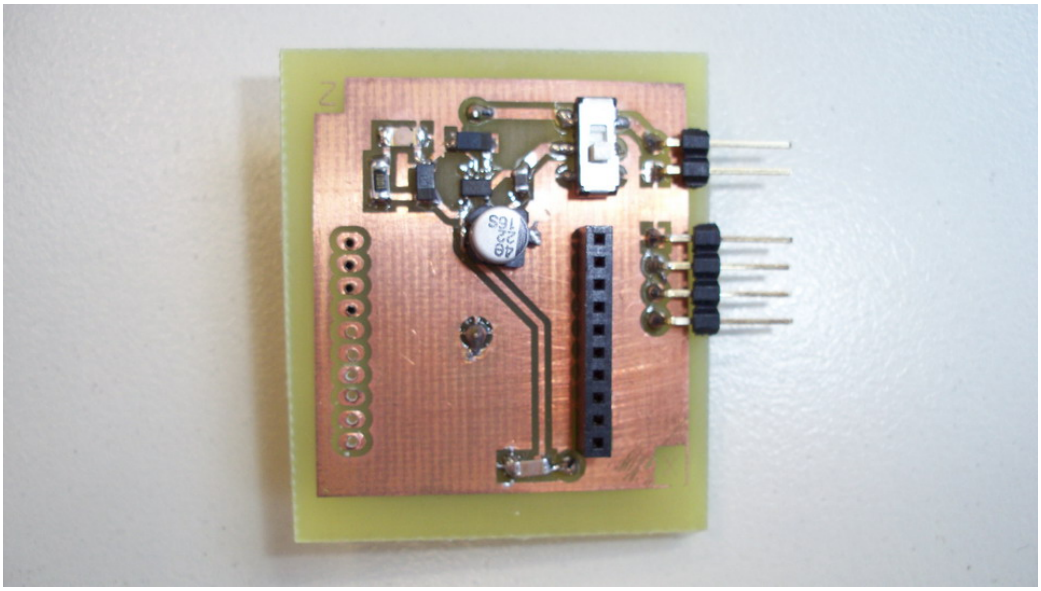
Οι μετρήσεις δεν παρουσίασαν κανένα πρόβλημα, το ίδιο και η αποστολή τους στον υπολογιστή. Το σύστημα λόγω της μειωμένης κατανάλωσής του μπορεί να λειτουργήσει για μεγάλο χρονικό διάστημα.

12.2 Προτάσεις για βελτίωση

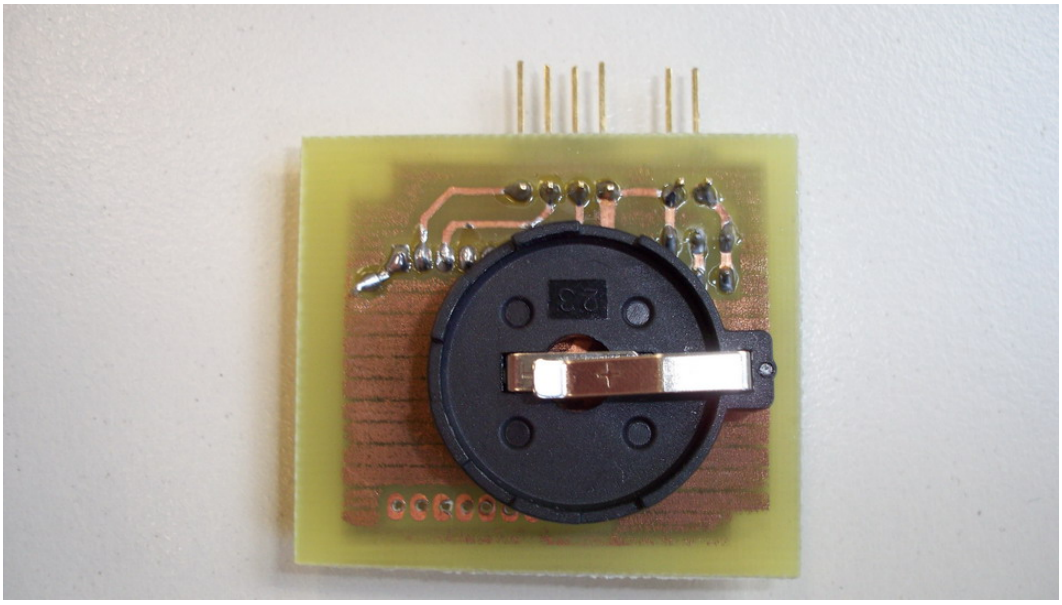
Αν και το σύστημα λειτουργεί πολύ ικανοποιητικά, υπάρχουν τρόποι να βελτιωθεί η λειτουργία του. Στην πλακέτα του μικροελεγκτή έχει προβλεφθεί και υπάρχει μια επιπλέον ακιδοσειρά για σύνδεση με την δεύτερη σειριακή θύρα του μικροελεγκτή. Στην θύρα αυτή μπορεί να τοποθετηθεί μια ασύρματη μονάδα (η οποία έχει κατασκευαστεί), η οποία θα στέλνει ασύρματα τα δεδομένα στον υπολογιστή και θα λαμβάνει εντολές. Οι εντολές θα δίνουν οδηγίες στον μικροελεγκτή πότε να αρχίσει την δειγματοληψία, για πόσο χρόνο να παίρνει δεδομένα, πότε να στείλει τα δεδομένα που πήρε, την συχνότητα δειγματοληψίας και πότε να μπει σε κατάσταση χαμηλής κατανάλωσης ενέργειας.



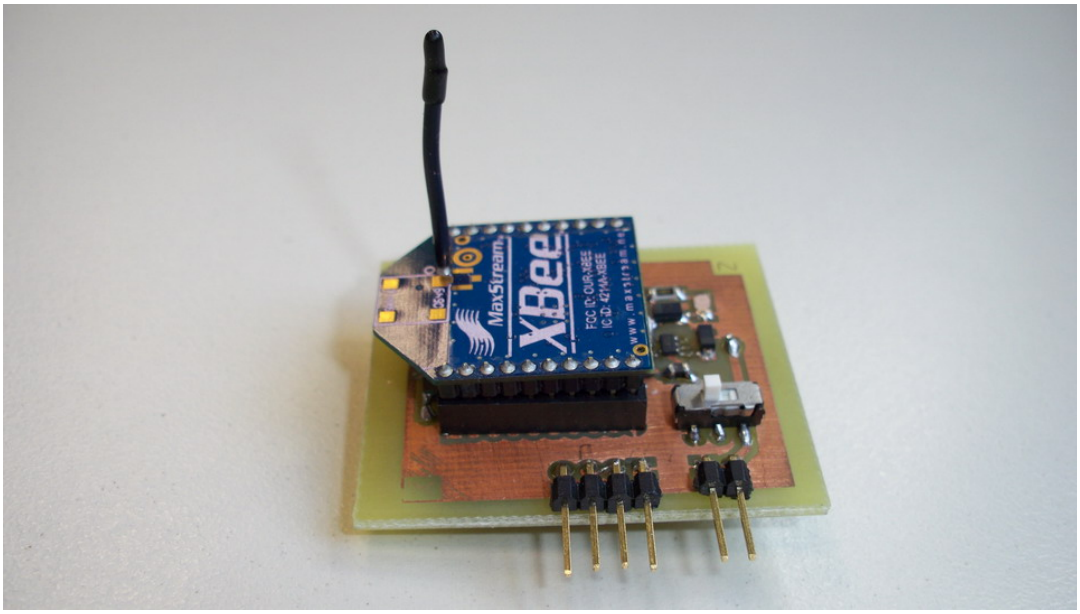
Σχήμα 12-1: Η ασύρματη μονάδα Xbee της Maxstream.



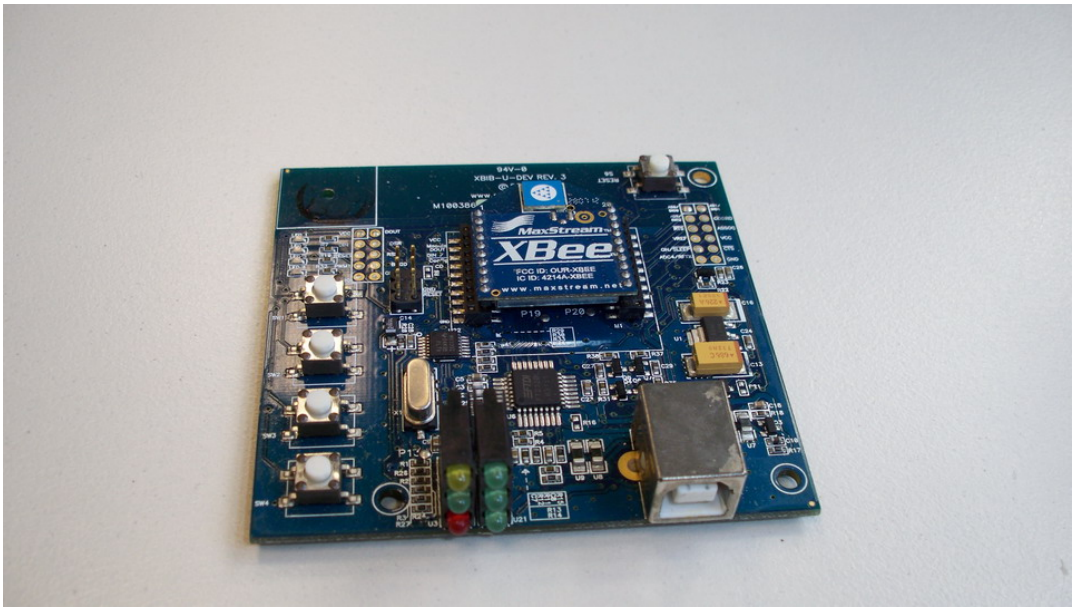
Σχήμα 12-2: Η πάνω όψη της πλακέτας ασύρματης επικοινωνίας.



Σχήμα 12-3: Η κάτω όψη της πλακέτας ασύρματης επικοινωνίας.



Σχήμα 12-4: Το υποσύστημα ασύρματης επικοινωνίας.



Σχήμα 12-5: Η πλακέτα της ασύρματης επικοινωνίας που συνδέεται στον υπολογιστή.

Επιπλέον μπορεί να δημιουργηθεί πρόγραμμα για τον υπολογιστή στο οποίο θα έχουμε ένα εύχρηστο γραφικό περιβάλλον, το οποίο θα μπορεί να στέλνει εντολές στον σύστμα του μικροελεγκτή, θα παίρνει τα δεδομένα με τις επιταχύνσεις και θα κάνει την επεξεργασία των σημάτων. Με αυτόν τον τρόπο θα μπορούμε να βλέπουμε τις γραφικές παραστάσεις σε πραγματικό χρόνο και να παίρνουμε στο τέλος τα στοιχεία που μας ενδιαφέρουν όπως: ταχύτητα βηματισμού, αριθμός βημάτων κ.α.

Επίσης θα μπορούσε να μειωθεί το μέγεθος του συστήματος και να τοποθετηθούν τα δύο υποσυστήματα σε μια πλακέτα για να μπορεί το σύστημα να γίνει ακόμα πιο εύχρηστο.

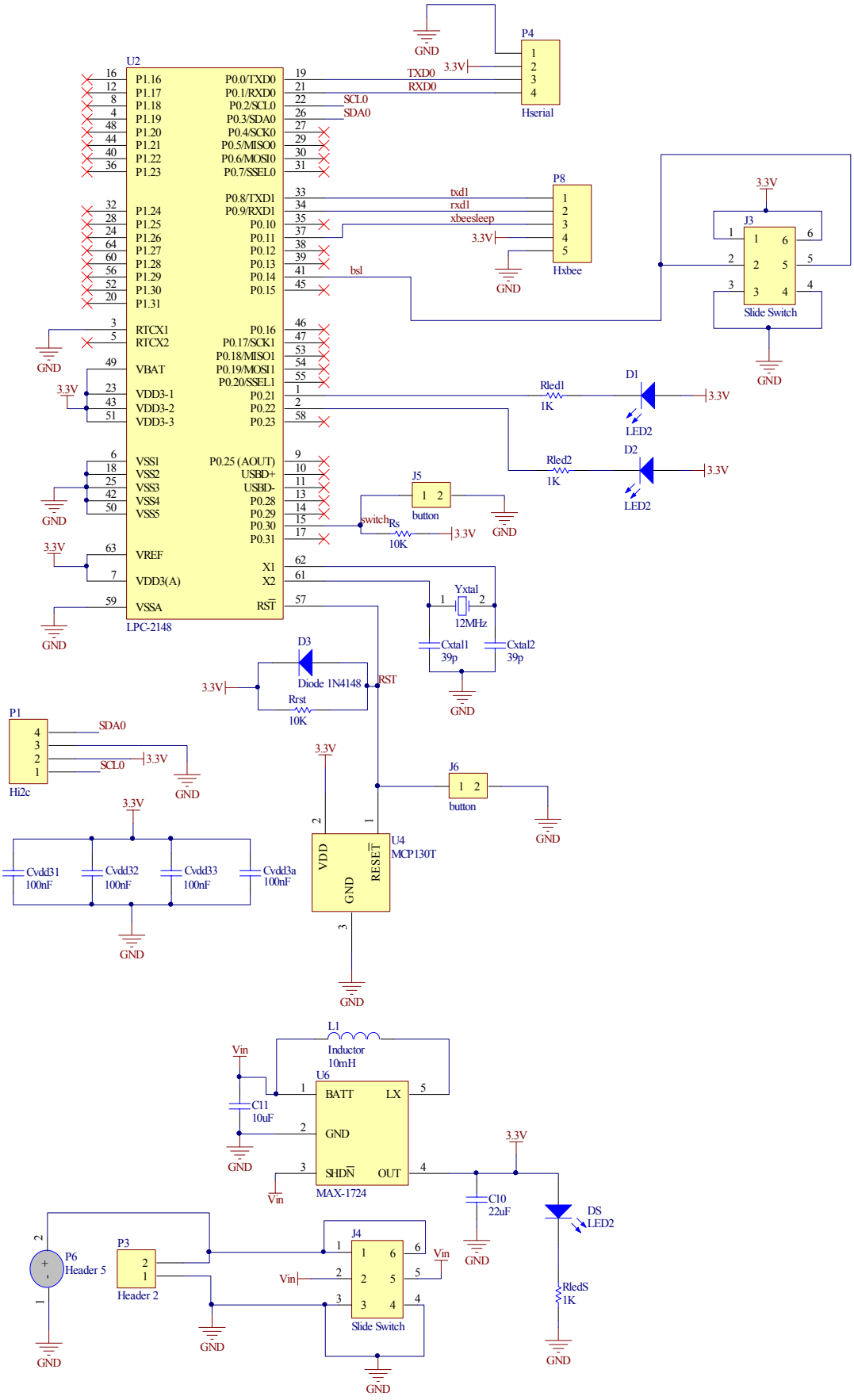
Τέλος, η κατανάλωση ισχύος θα μπορούσε να μειωθεί επιπλέον, προγραμματίζοντας τον μικροελεγκτή να “κοιμάται” όταν δεν διαβάζει επιταχύνσεις και όταν δεν στέλνει τα δεδομένα στον υπολογιστή. Το ολοκληρωμένο MMA7260 θα μπορούσε να μπαίνει σε κατάσταση χαμηλής κατανάλωσης όταν δεν χρειάζεται μέσω του sleep πιν που διαθέτει.

Βιβλιογραφία

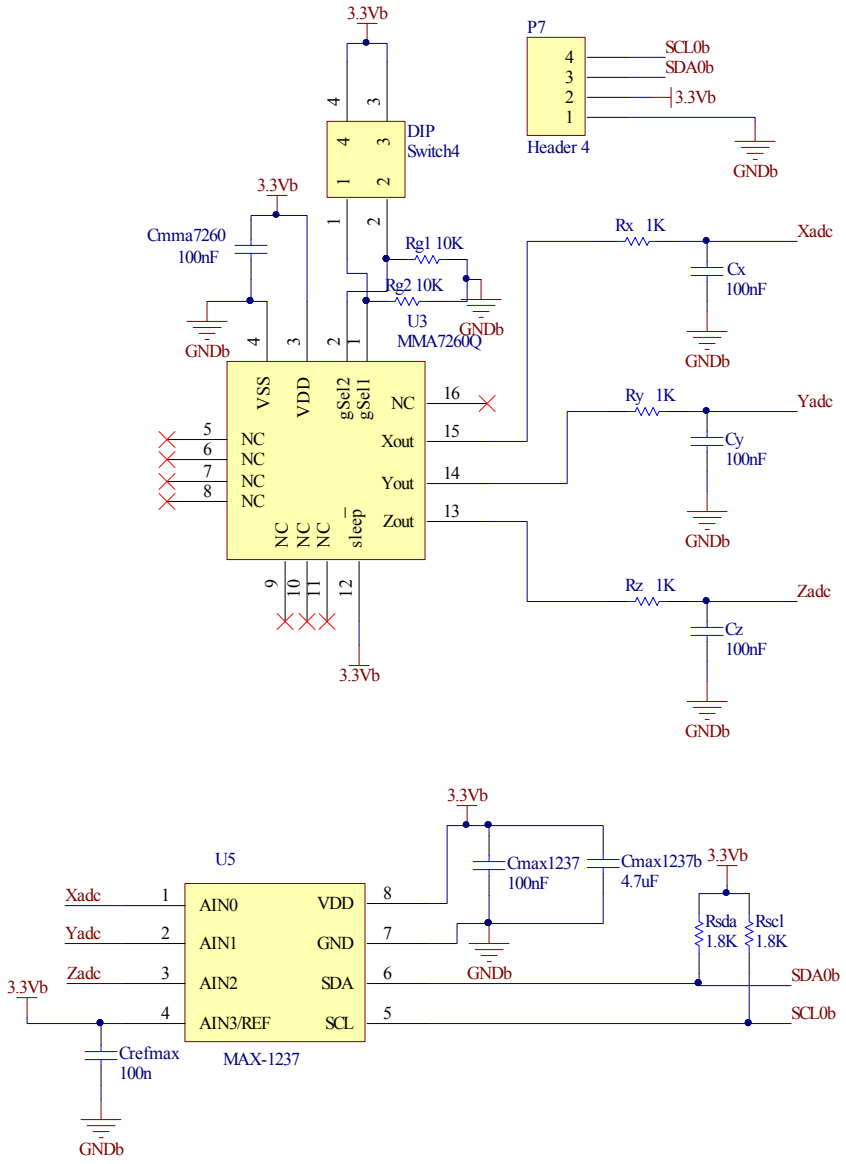
[1] Ι.Ν.Αβαριτσιώτης, Τεχνολογία αισθητήρων και μικροσυστημάτων, 2003.
[2] LPC214x User Manual UM10139, Philips Semiconductors
[3] ARM7TDMI datasheet ARM
[4] ARM System on chip architecture Steve Furber
[5] ARM System developers guide Andrew N. Sloss,Domonic Symes.
[6] GCC The complete reference Arthur Griffith
[7] Real-time processing with the Philips LPC ARM microcontroller; using GCC and the MicroC/OS-II RTOS. Philips 05: Project Number AR1803, 2005.
[8] J. J. Labrosse. Embedded Systems Building Blocks.
[9] B. Kernighan, D. Ritchie, "Η Γλώσσα Προγραμματισμού C".
[10]P. Aitken, B.L. Jones. Teach Yourself C in 21 Days, Fourth edition.
[11] Sedra, Smith. Microelectronics Circuits, Third edition.
[12] MMA7260QT Datasheet, Freescale Semiconductors
[13] MAX1237 Datasheet, Maxim
[14] MAX1724 Datasheet, Maxim
[15] MAX3232 Datasheet, Maxim
[16] XBee Datasheet, Digi
[17] MCP130T Datasheet, Microchip

ΠΑΡΑΡΤΗΜΑ
Α

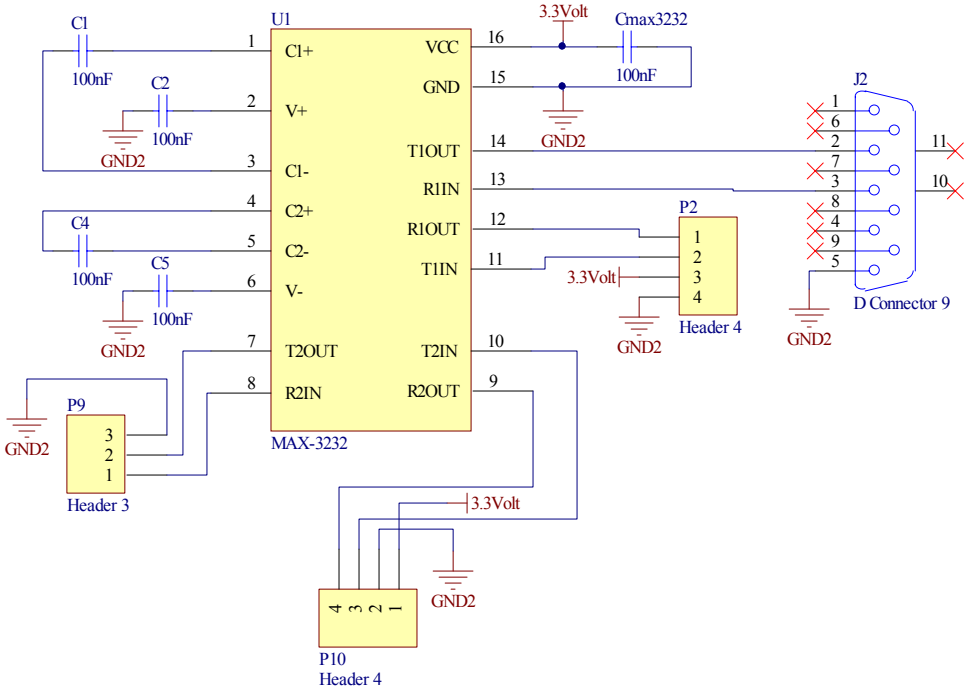
ΣΧΗΜΑΤΙΚΑ ΔΙΑΓΡΑΜΜΑΤΑ ΤΩΝ ΚΥΚΛΩΜΑΤΩΝ


U2
LPC-2148
P4
Hserial
TXD0
RXD0
SCL0
SDA0
P8
Hxbee
txd1
rxd1
xbeesleep
bsl
J3
Slide Switch
D1
LED2
Rled1
1K
D2
Rled2
J5
button
switch
Rs
10K
Yxtal
12MHz
Cxtal1
39p
Cxtal2
39p
D3
Diode 1N4148
Rrst
RST
P1
Hi2c
J6
U4
MCP130T
Cvdd31
100nF
Cvdd32
Cvdd33
Cvdd3a
L1
Inductor
10mH
U6
MAX-1724
BATT
LX
GND
SHDN
OUT
C11
10uF
C10
22uF
DS
LED2
RledS
P6
Header 5
P3
Header 2
J4
Slide Switch
Vin
3.3V
GND

Σχηματικό διάγραμμα του υποσυστήματος του μικροελεγκτή.



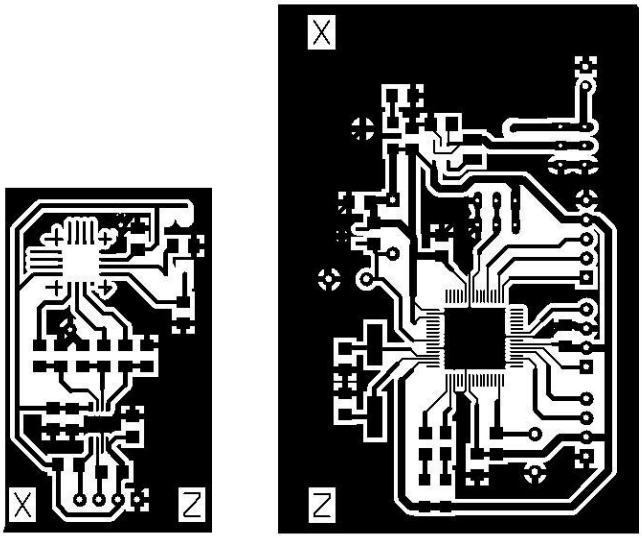
Σχηματικό διάγραμμα του υποσυστήματος του επιταχυνσιομέτρου.



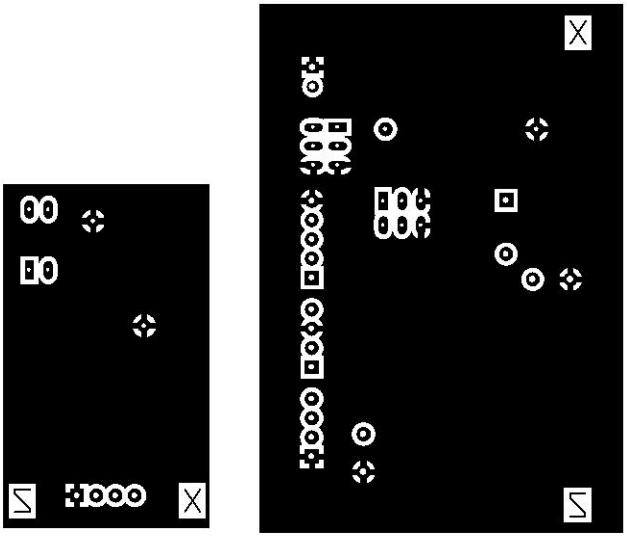
Σχηματικό διάγραμμα του υποσυστήματος του μετατροπέα τάσης RS232.

ΠΑΡΑΡΤΗΜΑ
Β

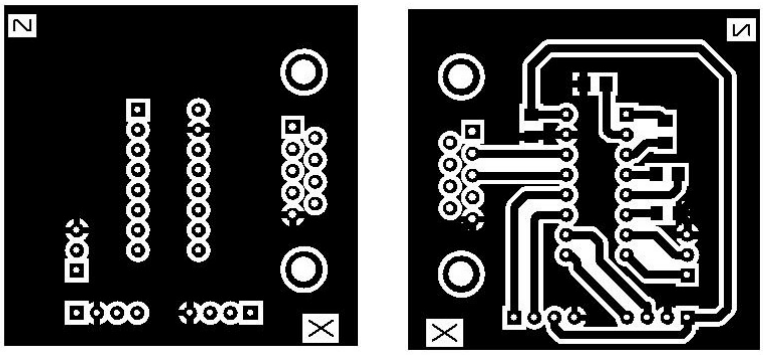
ΤΥΠΩΜΕΝΑ ΚΥΚΛΩΜΑΤΑ



Πάνω όψη των πλακετών των υποσυστημάτων επιταχυνσιομέτρου και μικροελεγκτή.



Κάτω όψη των πλακετών των υποσυστημάτων επιταχυνσιομέτρου και μικροελεγκτή.



Πάνω(αριστερά) και κάτω(δεξιά) όψη της πλακέτας του υποσυστήματος μετατροπής τάσης RS232.