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

Υλοποίηση αποκωδικοποιητή mp3 με τη χρήση του μικροελεγκτή AVR

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Β. Ρακόπουλος

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

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

Υλοποίηση αποκωδικοποιητή mp3 με τη χρήση του μικροελεγκτή AVR

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Β. Ρακόπουλος

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Ιουλίου 2005.

.....	.....	.....
Κ. Πεκμεστζή	Ον/μο Μέλος Δ.Ε.Π	Ον/μο Μέλος Δ.Ε.Π
Καθηγητής Ε.Μ.Π.	Ιδίοτητα Μέλους Δ.Ε.Π	Ιδίοτητα Μέλους Δ.Ε.Π

Αθήνα, Ιούλιος 2005

.....
Κωνσταντίνος Β. Ρακόπουλος

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

Σε όσους με στήριξαν
τον τελευταίο καιρό..

Περίληψη

Καθώς διατρέχουμε τον εικοστό πρώτο αιώνα ο κόσμος της τεχνολογίας έχει εισβάλλει "για τα καλά" στη καθημερινή ζωή. Όλο και περισσότερες ανέσεις προσφέρονται σήμερα στους ανθρώπους είτε για την διευκόλυνση , είτε για την διασκέδασή τους. Παρόλα αυτά ο σημερινός άνθρωπος ζητά όλο και περισσότερες ολοκληρωμένες ενσωματωμένες εφαρμογές για την χρησιμοποίηση τους στην καθημερινή ζωή. Κύριο συστατικό στην κατασκευή αυτών των εφαρμογών παίζουν οι μικροϋπολογιστές και μικροελεγκτές, οι οποίοι με τις τεράστιες δυνατότητες που προσφέρουν σήμερα, σε συνδυασμό με τον ελάχιστο όγκο που καταλαμβάνουν αλλά και την μικρή κατανάλωση τους, αποτελούν ιδανικά στοιχεία στην κατασκευή ενός ολοκληρωμένου φορητού ενσωματωμένου συστήματος.

Ένα τέτοιο σύστημα που έγινε πολύ δημοφιλές στις μέρες μας είναι ο φορητός MP3 αποκωδικοποιητής. Τα αρχεία MP3 με την πολύ καλή ποιότητα ήχου που προσφέρουν και το πολύ μικρό μέγεθος που απαιτούν σε μνήμη σε σχέση με άλλου είδους αρχεία ήχου, έγιναν στις μέρες μας επιλογή στην αναπαραγωγή ήχου. Ειδικότερα για τις ανταλλαγές αρχείων μέσω Internet όπου το εύρος ζώνης περιορίζει πολύ τις δυνατότητες για ανταλλαγή μεγάλων αρχείων τα MP3 αποτελούν πρώτη επιλογή. Συνεπώς η δημιουργία ενός φορητού συστήματος που μπορεί να αναπαράγει αυτά τα αρχεία είναι πολύ χρήσιμη στις μέρες μας καθώς έτσι μπορεί οποιοσδήποτε να το χρησιμοποιήσει για την ψυχαγωγία του ενώ κινείται.

Σκοπός της εργασίας είναι η αναλυτική περιγραφή της κατασκευής ενός φορητού συστήματος που αποκωδικοποιεί και αναπαράγει MP3 αρχεία. Εν προκειμένω, παρουσιάζονται αναλυτικά ο μικροελεγκτής ATmega32 της ATMEL που χρησιμοποιήθηκε καθώς και όλα τα περιφερειακά που χρησιμοποιήθηκαν για την επίτευξη του στόχου. Έτσι περιγράφονται αναλυτικά η συσκευή αποθήκευσης που χρησιμοποιήθηκε, το σύστημα αρχείων που χρησιμοποιήθηκε, η συσκευή για την απεικόνιση των τίτλων των τραγουδιών καθώς και ο αποκωδικοποιητής των MP3. Ακόμα εξηγείται με λεπτομέρεια το πρόγραμμα που γράφτηκε για αυτή την εφαρμογή ενώ δίνεται και ακριβής περιγραφή του hardware και του τρόπου σύνδεσης του. Τέλος παρουσιάζονται τα αποτελέσματα που επιτεύχθηκαν και μία πρόβλεψη για την φορητότητα του συστήματος και της κατανάλωσης ισχύος που απαιτεί.

Λέξεις-κλειδιά

AVR, Μικροελεγκτής, Multimedia κάρτα, Περιφερειακά, Σύστημα αρχείων FAT16, SPI, I2C, STA013, MP3, bitrate, ATmega32, LCD οθόνη, Μετατροπέας ψηφιακού σήματος σε αναλογικό, Αποκωδικοποιητής, Codevision AVR

Abstract

As we run through the twentieth first century the world of technology has entered "for good" in the daily life. Always more comforts are offered today in the persons or for their facilitation, or for their amusement. Nevertheless the current person asks always for more completed incorporated applications for utilisation in the daily life. Main component in the manufacture of these applications are the microcomputers and microcontrollers, which with the enormous capabilities that they offer today, in combination with the minimal volume that occupy and also with their small consumption, constitute ideal elements in the manufacture of completed portable incorporated system. Such a system that became very popular in our days is the portable MP3 decoder.

MP3 files with the very good quality of sound that offer and also the very small size that require in memory concerning other file types of sound, became in our days very good choice in the reproduction of sound. More specifically for the exchanges of files via Internet where the bandwidth limits a lot the possibilities for exchange of big files the MP3 files constitute first choice. Consequently the creation of portable system that can play these files is very useful in our days so can anyone use it for his entertainment while he is moving.

Aim of this work is the analytic description of manufacture of portable system that decodes and plays MP3 files. In this respect, they are presented analytically the microcontroller ATmega32 of ATMEL that was used as well as all peripheral that were used for the achievement of this objective. Thus they are described analytically the media of storage that was used, the system of files, the media for the display of titles of the songs as well as the decoder of MP3. Furthermore we explain with detail the program that was written for this application while is given also precise description of hardware and its way of connection. Finally there are presented the results that were achieved also a forecast for the portability of system and its power consumption.

Key-Words

AVR, Microcontroller, Multimedia card, Peripherals, System of files, FAT16, SPI, I2C, STA013, MP3, bitrate, ATmega32, LCD screen, Digital to Analog Converter, Decoder, Codevision AVR

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον καθηγητή μου κύριο Κιαμάλ Ζ. Πεκμεστζή, Καθηγητή Ε.Μ.Π. Επίσης θα ήθελα να ευχαριστήσω τον Κωνσταντίνο Γκότση για την βοήθεια που μου πρόσφερε κατά την διάρκεια της υλοποίησης αυτής της εργασίας. Τέλος θα ήθελα να ευχαριστήσω τους φίλους μου και την οικογένειά μου για την υποστήριξη που μου παρείχαν κατά την διάρκεια των τελευταίων μηνών.

Πίνακας Περιεχομένων

Πίνακας Πινάκων

Πίνακας Εικόνων

Λίστα ακρωνύμιων

AVR: Advanced Virtual RISC
RISC : Reduced Instruction Set Computing
MIPS : Million Instruction Per Second
SRAM : Static Random Access Memory
SP : Stack Pointer
SPI : Serial Peripheral Interface
MOSI : Master-Output Slave-Input
MISO: Master-Input Slave-Output
USART : Universal Synchronous Asynchronous Receiver Transmitter
TWI : Two Wire Interface
ADC : Analog to Digital Converter
DAC : Digital to Analog Converter
RS-MMC : Reduced Size MultiMedia Card
CRC : Cyclic Redundancy Code
LCD : Liquid Crystal Display
I2C : Intel Integrated Circuit
PCM : Pulse Code Modulation
MSB : Most Significant Bit
LSB : Least Significant Bit
FAT : File Allocation Table
MBR : Master Boot Record
VBR : Volume Boot Record

Κεφάλαιο 1

Εισαγωγή

1.1 – Εισαγωγή – Λίγα λόγια για τα .MP3 αρχεία

Σήμερα είναι αναγκαία η χρησιμοποίηση τεχνικών για την συμπίεση αρχείων ήχου. Αυτό συμβαίνει λόγω των αναγκών για εξοικονόμηση χώρου στα μέσα αποθήκευσης όσο και λόγω του περιορισμένου bandwidth του διαδικτύου.

Στις μέρες μας χρησιμοποιούνται πολλά είδη συμπίεσης για τα αρχεία ήχου. Μία από τις πιο δημοφιλείς είναι τα αρχεία .MP3. Τα MP3 είναι ένα σχήμα συμπίεσης βασισμένο σε απώλειες. Παρέχει μια αντιπροσωπευτική διαμόρφωση του κώδικα PCM ακουστικών στοιχείων σε ένα πολύ μικρότερο μέγεθος με την απόρριψη των στοιχείων που θεωρούνται "λιγότερο σημαντικά" στην ανθρώπινη ακρόαση. (Αυτό είναι παρόμοιο στην έννοια με τη συμπίεση JPEG για τις εικόνες.) Διάφορες τεχνικές υιοθετούνται στα MP3 για να καθορίσουν ποια στοιχεία του ήχου μπορούν να απορριφθούν. Στα MP3 ο ήχος μπορεί να συμπιεστεί με διαφορετικά ποσοστά δυαδικών ψηφίων (bit rates), που παρέχουν μια σειρά ανταλλαγών μεταξύ του μεγέθους των αρχείων και της ποιότητας ήχου.

Πίνακας 1.1: Αντιστοιχία Bit Rates και ποιότητας ήχου

Bit rates	Ποιότητα ήχου
4 Kbps	Ελάχιστη απαίτηση για αναγνώριση φωνής
8 Kbps	Ποιότητα τηλεφώνου
32 Kbps	MW ποιότητα
96 Kbps	FM ποιότητα
128 Kbps	CD ποιότητα

Η Mpeg-1 layer 2 κωδικοποίηση άρχισε ως ψηφιακό ακουστικό πρόγραμμα ραδιοφωνικής μετάδοσης (DAB) που αρχίζει από την Fraunhofer Society. Αυτό το έργο χρηματοδοτήθηκε από την Ευρωπαϊκή Ένωση ως μέρος του ερευνητικού προγράμματος EUREKA όπου ήταν γνωστό ως EE- 147.

Το EE- 147 λειτούργησε από το 1987 ως το 1994. Το 1991, υπήρξαν δύο προτάσεις διαθέσιμες: Musicam (γνωστό ως στρώμα II) και ASPEC (Adaptive Spectral Perceptual Entropy Coding) με ομοιότητες με το MP3. Το Musicam επιλέχτηκε λόγω της απλότητάς του.

Μια ομάδα εργασίας γύρω από τους Karl Heinz Brandenburg και Jürgen Herre πήρε ιδέες από τα Musicam και ASPEC, πρόσθεσε μερικές από τις ιδέες τους και δημιούργησε το MP3, το οποίο είχε ως σκοπό να επιτύχει την ίδια ποιότητα στα 128 Kbps με το MP2 στα 192 Kbps. Και οι δύο αλγόριθμοι οριστικοποιήθηκαν το 1992 ως τμήμα του Mpeg-1, η πρώτη τυποποιημένη ακολουθία από MPEG, που οδήγησε στη διεθνή τυποποίηση ISO, ενώ η περαιτέρω εργασία για τον ήχο MPEG οριστικοποιήθηκε το 1994 ως τμήμα της δεύτερης ακολουθίας Mpeg, Mpeg-2.

Η αποδοτικότητα συμπίεσης των -με απώλειες κωδικοποιητών- συμπίεσης καθορίζεται χαρακτηριστικά από το bit rate επειδή το ποσοστό συμπίεσης εξαρτάται από τον αριθμό των bits και το ποσοστό δειγματοληψίας του σήματος εισαγωγής. Εντούτοις, υπάρχουν συχνά δημοσιευμένα ποσοστά συμπίεσης που χρησιμοποιούν τις παραμέτρους του CD ως αναφορά (44,1 kHz, 2 κανάλια σε 16 μπιτ ανά κανάλι ή 2x16 bit). Μερικές φορές οι ψηφιακές παράμετροι SP ακουστικών ταινιών (DAT) χρησιμοποιούνται (48 kHz,

2x16 bit). Οι αναλογίες συμπίεσης για αυτήν την αναφορά είναι υψηλότερες, το οποίο καταδεικνύει το πρόβλημα του όρου "αναλογία συμπίεσης" για τους -με απώλειες κωδικοποιητές-.

Ο Karl Heinz Brandenburg χρησιμοποίησε μια καταγραφή του CD ,του τραγουδιού της Suzanne Vega, "Tom's Dinner" ως πρότυπό του για το MP3 αλγόριθμο συμπίεσης. Αυτό το τραγούδι επιλέχτηκε λόγω της μαλακότητας και της απλότητάς του, που καθιστούν ευκολότερο να ακούσει τις ατέλειες με το σχήμα συμπίεσης κατά τη διάρκεια των playbacks.

Τα ποσοστά συμπίεσης που προσφέρει ο αλγόριθμος συμπίεσης MP3 φαίνονται στον παρακάτω πίνακα:

Πίνακας 1.2: Αντιστοιχία Bit Rates και ποσοστού συμπίεσης

Bit Rate	Ποσοστό συμπίεσης
384 Kbps	4 : 1
192 – 256 Kbps	8 : 1 έως 6 : 1
112 - 128 Kbps	12 : 1 έως 10 : 1

Βλέπουμε ότι σε μία πολύ καλή ποιότητα ήχου των 128 Kbps η συμπίεση MP3 προσφέρει μία μεγάλη εξοικονόμηση μνήμης κάτι που κάνει τα αρχεία MP3 ιδεατά για φορητές εφαρμογές αναπαραγωγής ήχου όπου η μνήμη είναι περιορισμένη.

1.2 – Σκοπός και οργάνωση της εργασίας

Σκοπός της εργασίας είναι η αναλυτική περιγραφή της κατασκευής ενός φορητού MP3 αποκωδικοποιητή. Για την υλοποίηση του αποκωδικοποιητή επιλέχθηκε η χρησιμοποίηση εξωτερικού hardware που κάνει την αποκωδικοποίηση των αρχείων MP3 και μικροελεγκτή για τον έλεγχο όλων των περιφερειακών. Η λύση αυτή κρίνεται σαφώς ανώτερη από άλλες τακτικές σχεδίασης του συστήματος λόγω της ευκολίας που προσφέρει στον σχεδιασμό και τον προγραμματισμό ενώ παράλληλα το κόστος είναι σχετικά μικρό.

Προκειμένου να γίνει σαφής στο αναγνώστη ο τρόπος υλοποίησης του αποκωδικοποιητή εξετάζονται μέσα στο κείμενο αναλυτικά ο μικροελεγκτής που χρησιμοποιήθηκε, τα περιφερειακά που συνδέθηκαν με το μικροϋπολογιστή, η σύνδεση που σχεδιάστηκε και υλοποιήθηκε καθώς και το πρόγραμμα που γράφτηκε αλλά και τα εργαλεία λογισμικού που χρησιμοποιήθηκαν.

Αναλυτικά ο τρόπος οργάνωσης της εργασίας παρουσιάζεται παρακάτω:

Στο **κεφάλαιο 2** γίνεται αναλυτική περιγραφή του μικροελεγκτή ATmega32 της ATMEL που χρησιμοποιήθηκε. Γίνεται περιγραφή όλων των χαρακτηριστικών γνωρισμάτων που έχει και σύγκριση με άλλες επιλογές που θα μπορούσαν να γίνουν ώστε να δειχθεί η αξία της επιλογής του συγκεκριμένου μικροελεγκτή.

Στο **κεφάλαιο 3** γίνεται μνεία σε όλα τα περιφερειακά που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας. Αναλύεται ο τρόπος λειτουργίας τους και των κυριότερων χαρακτηριστικών τους.

Στο **κεφάλαιο 4** περιγράφεται το σύστημα αρχείων που χρησιμοποιήθηκε για την αποθήκευση των αρχείων MP3. Ακόμα γίνεται σύντομη αναφορά στα υπάρχοντα συστήματα αρχείων έτσι ώστε να δοθεί η δυνατότητα στον αναγνώστη να έχει μία γενικότερη εικόνα πάνω στο θέμα.

Στο **κεφάλαιο 5** περιγράφεται αναλυτικά ο τρόπος σύνδεσης των περιφερειακών με τον μικροελεγκτή. Ακόμα παρατίθενται το σχηματικό διάγραμμα του κυκλώματος αλλά και το layout του τυπωμένου κυκλώματος που κατασκευάστηκε.

Στο **κεφάλαιο 6** αναλύεται το κυρίως πρόγραμμα που γράφτηκε αλλά και τα επιμέρους προγράμματα οδήγησης του κάθε περιφερειακού. Εξηγούνται οι χρονισμοί που δημιουργήθηκαν για την σωστή λειτουργία του κυκλώματος.

Στο **παράρτημα Α** παρατίθεται το πρόγραμμα που γράφτηκε σε γλώσσα C πλήρως σχολιασμένο.

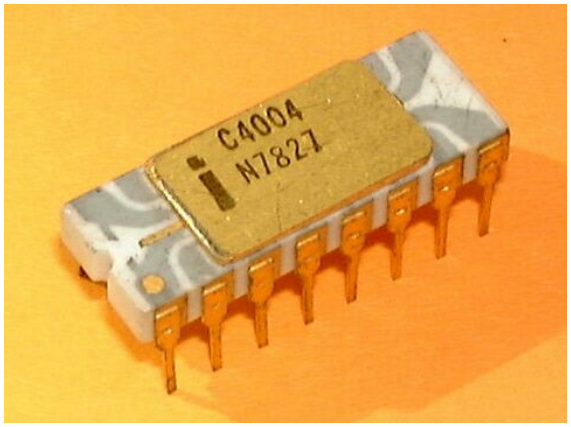
Στο **παράρτημα Β** περιγράφεται το CodeVision AVR και δίνεται ένα απλό παράδειγμα προγραμματισμού του ώστε να γίνει κατανοητό στον χρήστη.

Κεφάλαιο 2

Ο μικροελεγκτής AVR

2.1 – Ιστορική αναδρομή στους μικροϋπολογιστές

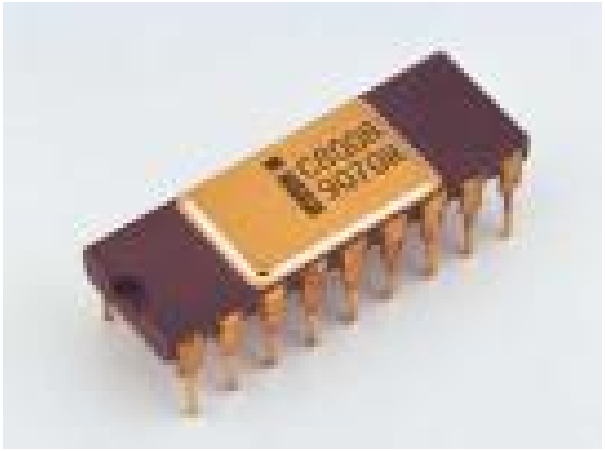
Ο πρώτος μικροεπεξεργαστής αναπτύχθηκε από αυτό που ήταν τότε μια μικρή επιχείρηση αποκαλούμενη Intel (συντόμευση για την ενσωματωμένη ηλεκτρονική – INTegrated ELectonics) στις αρχές της δεκαετίας του '70. Ο πελάτης, μια ιαπωνική επιχείρηση αποκαλούμενη Busicon, αρνήθηκε να αγοράσει το chipset και η Intel, που βρέθηκε αντιμέτωπη με ένα κόστος ανάπτυξης και κανένα πελάτη, αποφάσισε να εμπορευτεί το chipset ως microprocessing "γενικού σκοπού" σύστημα για τη χρήση στις εφαρμογές όπου τα ψηφιακά τσιπ λογικής είχαν χρησιμοποιηθεί. Το chipset ήταν μια επιτυχία και μέσα σε έναν μικρό χρονικό διάστημα η Intel ανάπτυξε έναν γενικού σκοπού μικροεπεξεργαστή 4 μπιτ που ονομάστηκε 4004.



Εικόνα 2.1 : Ο πρώτος μικροεπεξεργαστής 4004 της Intel

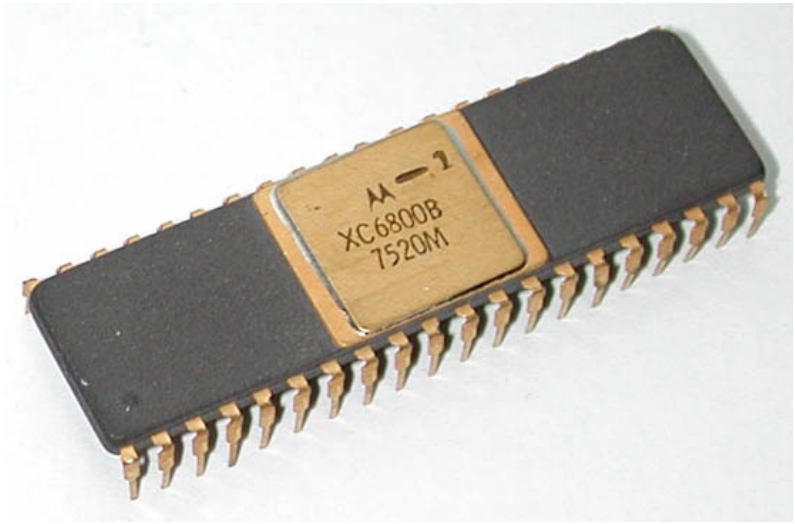
Ταυτόχρονα σχεδόν αναπτύχθηκαν άλλοι δύο τετράμπιτοι επεξεργαστές ο TMS100 της TI και ο επεξεργαστής MP944 που χρησιμοποιήθηκε από το αμερικάνικο ναυτικό για τα μαχητικά αεροπλάνα.

Ο 4004 ακολουθήθηκε αργότερα από τον 8008, ο παγκόσμια πρώτος οκτάμπιτος μικροεπεξεργαστής. Αυτοί οι επεξεργαστές είναι οι πρόδρομοι των πολύ επιτυχημένων Intel 8080, Zilog Z80, και τους ακόλουθους οκτάμπιτους επεξεργαστές της Intel.

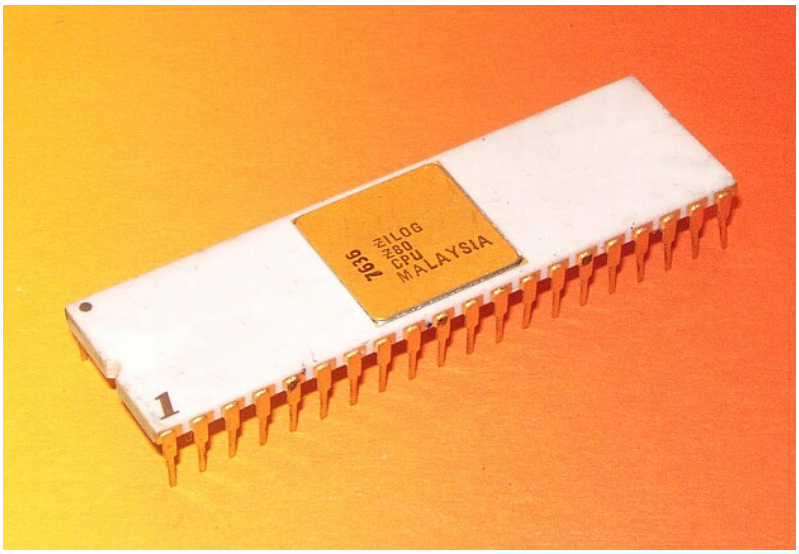


Εικόνα 2.2 : Ο πρώτος οκτάμπιτος, ο 8008 της Intel

Η ανταγωνιστική αρχιτεκτονική του Motorola 6800 κλωνοποιήθηκε και βελτιώθηκε στην τεχνολογία 6502 MOS, που οδήγησε στον Z80 που είχε μεγάλη δημοτικότητα κατά τη διάρκεια της δεκαετίας του '80. Και ο Z80 και ο 6502 επικεντρώθηκαν στο χαμηλό γενικό κόστος, μέσω ενός συνδυασμού μικρής συσκευασίας, τις απλές απαιτήσεις των διαδρόμων των υπολογιστών, και το συνυπολογισμό των στοιχείων κυκλώματος που θα έπρεπε κανονικά να παρασχεθούν σε ένα χωριστό τσιπ (παραδείγματος χάριν, ο Z80 περιέλαβε έναν ελεγκτή μνήμης). Ήταν αυτά τα χαρακτηριστικά γνωρίσματα που επέτρεψαν στον προσωπικό υπολογιστή την "επανάσταση" στις αρχές της δεκαετίας του '80, παραδίδοντας τελικά τις ημι-χρησιμοποιήσιμες μηχανές που πωλούνταν για $99.



Εικόνα 2.3 : Ο 6800 της MOTOROLA



Εικόνα 2.4 : Ο συνεχιστής του 6800 – Zilog Z-80

Η Motorola ανέτρεψε ολόκληρο τον οκτάμπιτο κόσμο με την εισαγωγή του MC6809, ένα από τα ισχυρότερα σχέδια επεξεργαστή - και επίσης ένα από τον πιό σύνθετα σχέδια λογικής που έγιναν ποτέ στην παραγωγή οποιουδήποτε μικροεπεξεργαστή. Το Microcoding αντικατέστησε την hardware λογική σχεδίαση επειδή οι απαιτήσεις σχεδίου έγιναν πάρα πολύ σύνθετες για την λογική σχεδίαση.

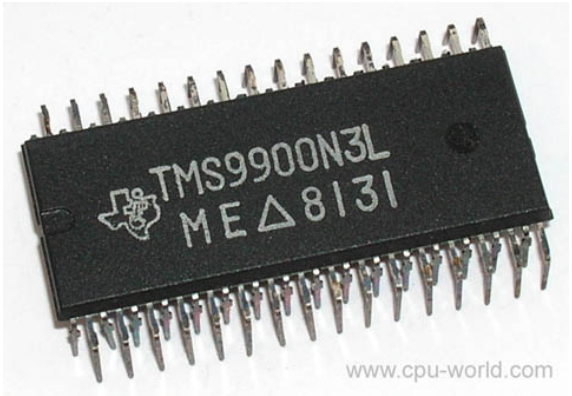
Ένας άλλος πρόωρος οκτάμπιτος μικροεπεξεργαστής ήταν το Signetics 2650, το οποίο απόλαυσε μια σύντομη προσέλκυση του ενδιαφέροντος λόγω στην καινοτόμο και ισχυρή αρχιτεκτονική του.

Ένας δημιουργικός μικροεπεξεργαστής στον κόσμο της διαστημικής πτήσης ήταν RCA 1802 της RCA που χρησιμοποιήθηκε στους Voyager και Viking της NASA της δεκαετίας του '70, και στην παρακολούθηση του Γαλιλαίου στο Δία (που εκτοξεύθηκε το 1989 και έφτασε το 1995). Το CDP1802 χρησιμοποιήθηκε επειδή μπόρεσε να έχει πολύ μικρή κατανάλωση και επειδή η διαδικασία παραγωγής του εξασφάλισε πολύ καλύτερη προστασία ενάντια στην κοσμική ακτινοβολία και τις ηλεκτροστατικές αποφορτίσεις από αυτή οποιουδήποτε άλλου επεξεργαστή της εποχής κατά συνέπεια, τα 1802 λέγεται ότι είναι ο πρώτος ανθεκτικός σε ακτινοβολία- μικροεπεξεργαστής.

Ο πρώτος δεκαεξάμπιτος μικροεπεξεργαστής multi-chip ήταν ο IMP- 16 της National, που εισήχθη στις αρχές του 1973. Μια οκτάμπιτη έκδοση του chipset ανακοίνωσε η NATIONAL το 1974 ως IMP-8. Το 1975, η NATIONAL εισήγαγε το πρώτο δεκαεξάμπιτο single-chip μικροεπεξεργαστή, τον PACE, ο οποίος ακολουθήθηκε αργότερα από μια nmos έκδοση, το INS8900.

Άλλοι πρόωροι δεκαεξάμπιτοι μικροεπεξεργαστές multi-chip περιλαμβάνουν τον ένα που χρησιμοποιείται από την Digital Equipment Corporation (Δεκέμβριος) στο LSI- 11 board OEM και το συσκευασμένο μίνι-υπολογιστή PDP 11/03, καθώς και τον Fairchild Semiconductor MicroFlame 9440, και τα δύο από τα οποία εισήχθησαν στο χρονικό πλαίσιο του 1975 έως 1976.

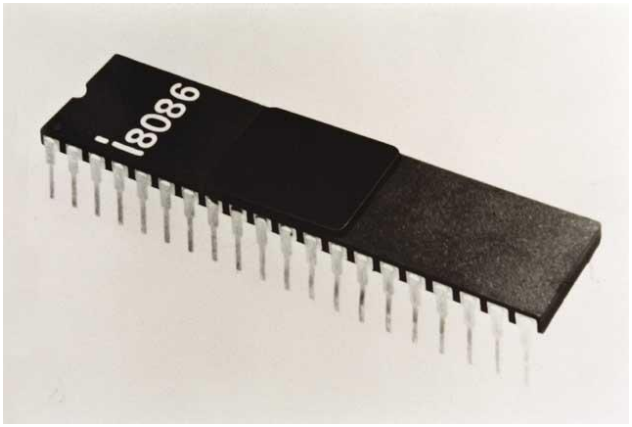
Ο πρώτος single-chip δεκαεξάμπιτος μικροεπεξεργαστής ήταν ο TMS 9900 της TI, το οποίο ήταν επίσης συμβατός με τη έκδοση TI 990 μίνι-υπολογιστών της. Τα 9900 χρησιμοποιήθηκαν στο μίνι-υπολογιστή TI 990/4, τον προσωπικό υπολογιστή TI- 99/4A, και την TM990 boards μικροϋπολογιστών OEM. Το τσιπ συσκευάστηκε σε μια μεγάλη κεραμική συσκευασία DIP - 64pin, ενώ οι περισσότεροι οκτάμπιτοι μικροεπεξεργαστές όπως ο Intel 8080 χρησιμοποίησαν την πιό κοινή, μικρότερη, και λιγότερο ακριβή πλαστική DIP - 40pin. Ένα τσιπ συνέχεια, το TMS 9980, σχεδιάστηκε για να ανταγωνιστεί με την Intel 8080,και είχε το πλήρες TI990, δεκαεξάμπιτες εντολές και χρησιμοποίησε πλαστικές 40pin συσκευασίες, μεταφορά δεδομένων 8 μπιτ τη φορά, αλλά θα μπορούσαν μόνο να διευθυσιοδοτήσουν 16 KB. Ένα τρίτο τσιπ, το TMS 9995, ήταν ένα νέο σχέδιο. Η οικογένεια επεκτάθηκε αργότερα για να περιλάβει τα 99105 και τα 99110.



Εικόνα 2.5 : Ο πρώτος δεκαεξάμπιτος επεξεργαστής TMS9900

Η Intel ακολούθησε μια διαφορετική πορεία αναβαθμίζοντας το σχέδιο του 8080 στον δεκαεξάμπιτο Intel 8086, το πρώτο μέλος της x86 οικογένειας που χρησιμοποιούν οι περισσότεροι σύγχρονοι υπολογιστές. Η Intel εισήγαγε τον 8086 ως οικονομικώς αποδοτικό τρόπο μεταφοράς του λογισμικού από τον 8080, και πέτυχε κερδίζοντας την εμπιστοσύνη πολλών επιχειρήσεων με εκείνη την προϋπόθεση. Ακολουθώντας τους 8086 και 8088, η

Intel απελευθέρωσε τους 80186 ..80286 και, το 1985, τον τριανταδυάμπιτο 80386, παγιώνοντας την κυριαρχία την στην αγορά PC.



Εικόνα 2.6: Ο 8086 της INTEL –Η κυριαρχία της INTEL αρχίζει

Τα δεκαεξάμπιτα σχέδια ήταν στην αγορά μόνο για λίγο όταν άρχισαν να εμφανίζονται οι πλήρεις τριανταδυάμπιτες εφαρμογές.

Ο παγκόσμιος πρώτος single-chip τριανταδυάμπιτος μικροεπεξεργαστής ήταν ο AT&T Bell Labs BELLMAC-32A,, με τα πρώτα δείγματα το 1980, και γενική παραγωγή το 1982. Μετά από την αποστέρηση του AT&T το 1984, μετονομάστηκε σε WE 32000 (Western Electric), και είχε δύο γενεές συνέχισης, WE 32100 και WE 32200. Αυτοί οι μικροεπεξεργαστές χρησιμοποιήθηκαν στους μίνι-υπολογιστές AT&T 3B5 και 3B15 στο 3B2, τον πρώτο super microcomputer γραφείου. Όλα αυτά τα συστήματα έτρεξαν το αρχικό λειτουργικό σύστημα Unix των Bell Labs.

Ο διασημότερος των τριανταδυάμπιτων σχεδίων είναι ο MC68000, που εισάγεται το 1979. Ο 68K όπως ήταν ευρέως γνωστό, είχε τριανταδυάμπιτους καταχωρητές αλλά χρησιμοποίησε τις δεκαεξάμπιτες διαδρομές δεδομένων, και ένα δεκαεξάμπιτο εξωτερικό bus δεδομένων για να μειώσει τα pins. Η Motorola το περιέγραψε γενικά ως δεκαεξάμπιτο επεξεργαστή, αν και έχει σαφώς την τριανταδυάμπιτη αρχιτεκτονική. Ο συνδυασμός υψηλής ταχύτητας, μεγάλου χώρου αποθήκευσης (16 MB) και αρκετά χαμηλού κόστους το έκανε το δημοφιλέστερο CPU της κατηγορίας του.

Εικόνα 2.7 : Ο δημοφιλής το ’80 MC68000

Η Apple Lisa και η Macintosh χρησιμοποίησαν τα 68000, σε ένα πλήθος σχεδίων στα μέσα της δεκαετίας του '80, συμπεριλαμβανομένου του Atari ST και του Commodore Amiga.

Ο πρώτος τριανταδυάμπιτος μικροεπεξεργαστής της Intel ήταν το iAPX 432, το οποίο εισήχθη το 1981 αλλά δεν ήταν μια εμπορική επιτυχία. Είχε μια προηγμένη στην

ικανότητα-βασισμένη αντικειμενοστραφή αρχιτεκτονική, αλλά η κακή απόδοση ήταν καταδικαστική συγκρινόμενη με άλλες ανταγωνιστικές αρχιτεκτονικές όπως το Motorola 68000.

Η επιτυχία του Motorola με τα 68000 οδήγησε στο MC68010, το οποίο πρόσθεσε την υποστήριξη εικονικής μνήμης. Το MC68020, που εισήχθη το 1985 πρόσθεσε τα πλήρη τριανταδυάμπιτα bus δεδομένων και διευθύνσεων. Τα 68020 έγιναν σημαντικά δημοφιλή στη super microcomputer Unix αγορά, και πολλές μικρές επιχειρήσεις (π.χ. Altos, Charles River Data Systems) παρήγαγαν τα συστήματα desktop-μεγέθους. Μετά από αυτό με το MC68030, που πρόσθεσε το MMU στο τσιπ, η οικογένεια 68K έγινε ο επεξεργαστής για όλα που δεν έτρεχαν το DOS. Η συνεχής επιτυχία οδήγησε στο MC68040, το οποίο περιέλαβε ένα FPU για την καλύτερη απόδοση. Ο 68050 απέτυχε να επιτύχει τους στόχους απόδοσής του και δεν απελευθερώθηκε, και το ακόλουθο MC68060 απελευθερώθηκε σε μια αγορά που διαποτίστηκε από τα πολύ γρηγορότερα σχέδια RISC. Η οικογένεια 68K εξασθένισε από την αγορά υπολογιστή γραφείου στις αρχές της δεκαετίας του '90.

Άλλες μεγάλες επιχειρήσεις σχεδίασαν τα 68020. Σε ένα σημείο, υπήρξαν περισσότερα 68020s στον ενσωματωμένο εξοπλισμό από ότι Intel Pentiums σε PCs. Οι πυρήνες επεξεργαστών ColdFire είναι παράγωγα των σεβαστών 68020.

Κατά τη διάρκεια αυτής της περιόδου (νωρίς στη δεκαετία του '80), η NATIONAL εισήγαγε ένα πολύ παρόμοιο με δεκαεξάμπιτο pinout, τριανταδυάμπιτο μικροεπεξεργαστή αποκαλούμενο NS 16032 (αργότερα μετονομάζεται σε 32016), τη πλήρη τριανταδυάμπιτη έκδοση που ονομάζεται NS 32032, και μια γραμμή τριανταδυάμπιτων βιομηχανικών μικροϋπολογιστών OEM. Μέχρι τα μέσα της δεκαετίας του '80, η Sequent εισήγαγε τον πρώτο συμμετρικό υπολογιστή υπολογιστής-κατηγορίας πολυεπεξεργαστών (SMP) χρησιμοποιώντας το NS 32032. Αυτό ήταν ένα από τα σχέδια που λίγοι θέλουν, και εξαφανίστηκε προς το τέλος της δεκαετίας του '80.

Άλλα σχέδια περιέλαβαν το ενδιαφέρον Zilog Z8000, το οποίο έφθασε πάρα πολύ αργά στην αγορά για να σταθεί και εξαφανίστηκε γρήγορα. Προς το τέλος της δεκαετίας του '80, οι "πόλεμοι μικροεπεξεργαστών" άρχισαν να εξαφανίζουν μερικούς από τους μικροεπεξεργαστές με αποτέλεσμα την κυριαρχία της INTEL και των Pentium.

Εικόνα 2.8 : Ο Pentium 4

Αν και οι σε RISC αρχιτεκτονική βασισμένοι σχεδιασμοί ήταν η πρώτη συγκομιδή των επεξεργαστών 64 μπιτ πολύ πριν από τα τρέχοντα μικροτσίπ PC από την IBM, AMD και την Intel, τα 64 μπιτ άρχισαν να εφαρμόζονται στον υπολογιστή γραφείου το 2003 με επίσημες ανακοίνωση του AMD Opteron τον Απρίλιο, του AMD Athlon 64 το Σεπτέμβριο, του PowerPC G5 τον Ιούνιο και του Intel Xeon το 2004.

Με την εισαγωγή από την AMD του πρώτου προς τα πίσω συμβατού εξηντατετράμπιτου τσιπ Athlon 64 τον Σεπτέμβριο του 2003, που ακολουθείται από της Intel τον 64 μπιτ επεξεργαστή, εδραιώνεται η εποχή υπολογιστών γραφείου 64 μπιτ.

Εικόνα 2.9 : Νέα γενιά - Athlon 64 της AMD

Εικόνα 2.10 : Νέα γενιά – Itanium2 της Intel

Και οι δύο επεξεργαστές μπορούν να τρέξουν την κληρονομιά των 32 μπιτ εφαρμογών καθώς επίσης και το νέο λογισμικό 64 μπιτ. Με τα WINDOWS XP και Linux 64 μπιτ που τρέχουν σε 64 μπιτ, το λογισμικό επίσης συνδέεται για να χρησιμοποιήσει την πλήρη δύναμη τέτοιων επεξεργαστών. Στην πραγματικότητα η κίνηση προς τα 64 μπιτ είναι περισσότερο από ακριβώς μια αύξηση στο μέγεθος καταχωρητών από τα 32 bits δεδομένου ότι περιλαμβάνει επίσης μια μικρή αύξηση σε ποσότητα καταχωρητών για τα σχέδια CISC.

Η κίνηση προς τα 64 μπιτ από τους PowerPC επεξεργαστές είχε προοριστεί δεδομένου ότι οι επεξεργαστές που σχεδιάστηκαν στις αρχές της δεκαετίας του '90 δεν είχαν μια σημαντική αιτία του ασυμβιβάστου. Οι υπάρχοντες καταχωρητές ακέραιων αριθμών επεκτείνονται όπως και όλες οι σχετικές διαδρομές στοιχείων αλλά από καιρό η την αρχιτεκτονική 32 bits σχεδιάστηκε με την κινητή υποδιαστολή και τις διανυσματικές μονάδες να λειτουργούν σε ή επάνω από 64 μπιτ για αρκετά έτη. Ακόμα από την αρχιτεκτονική των 32 bits κανένας νέος καταχωρητής γενικού σκοπού προστίθεται έτσι οποιαδήποτε αντιδικία για την μη χρησιμοποίηση του τρόπου 64 μπιτ όπου διαθέσιμος είναι ελάχιστη.

2.2 – Μικροελεγκτές – Μία λίγο διαφορετική ιστορία

Ένας μικροελεγκτής είναι ένα υπολογιστής – σε ένα - chip που βελτιστοποιείται για να ελέγξει τις συσκευές. Είναι ένας τύπος μικροεπεξεργαστή που υποστηρίζει την αυτάρκεια και την οικονομική αποτελεσματικότητα, σε αντίθεση με έναν γενικής χρήσης μικροεπεξεργαστή, το είδος που χρησιμοποιείται σε ένα PC. Ένας χαρακτηριστικός μικροελεγκτής περιέχει όλη τη μνήμη και τις I/O γραμμές που απαιτούνται, ενώ ένας μικροεπεξεργαστής γενικού σκοπού απαιτεί πρόσθετα τσιπ για να παρέχει αυτές τις απαραίτητες λειτουργίες.

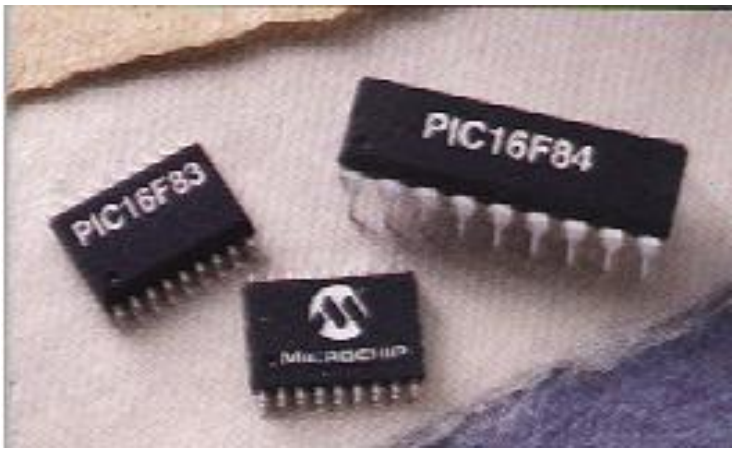
Οι μικροελεγκτές είναι κύριο συστατικό σε πολλά είδη ηλεκτρονικού εξοπλισμού. Είναι η μεγάλη πλειοψηφία όλων των τσιπ επεξεργαστών που πωλούνται. Πάνω από 50% είναι "απλοί" ελεγκτές, και άλλα 20% είναι πιό ειδικευμένοι επεξεργαστές ψηφιακών σημάτων (DSPs). Ένα χαρακτηριστικό σπίτι στο δυτικό κόσμο είναι πιθανό να έχει μόνο έναν ή δύο γενικής χρήσης μικροεπεξεργαστές αλλά κάπου μεταξύ μίας και δύο δωδεκάδων μικροελεγκτών. Μπορούν να βρεθούν σχεδόν σε οποιουσδήποτε τύπου ηλεκτρικής συσκευής, πλυντήρια ρούχων, φούρνους μικροκυμάτων, τηλέφωνα κ.λ.π.

Οι περισσότεροι μικροελεγκτές είναι σήμερα βασισμένοι στην αρχιτεκτονική Von Neumann, η οποία καθόρισε σαφώς τα τέσσερα βασικά συστατικά που απαιτούνται για ένα ενσωματωμένο σύστημα. Αυτά περιλαμβάνουν έναν πυρήνα CPU, τη μνήμη για το πρόγραμμα (μνήμη ROM ή FLASH), τη μνήμη για τις μεταβλητές (RAM), έναν ή περισσότερους timers (αυτόνομοι και watchdog), καθώς επίσης και τις I/O γραμμές για να επικοινωνήσουν με τις εξωτερικές περιφερειακές μονάδες και τους συμπληρωματικούς πόρους - όλο αυτό σε ένα ενιαίο ολοκληρωμένο κύκλωμα. Ένας μικροελεγκτής διαφέρει από ένα γενικής χρήσης τσιπ CPU επειδή το δεύτερο είναι γενικά αρκετά εύκολο να σταθεί σε έναν λειτουργών υπολογιστή, με το ελάχιστο των εξωτερικών τσιπ υποστήριξης. Η ιδέα είναι ότι ο μικροελεγκτής θα τοποθετηθεί στη συσκευή για έλεγχο, συνδέεται στην τροφοδοσία και σε οποιεσδήποτε πληροφορίες χρειάζεται, και αυτό είναι όλο.

Εικόνα 2.11 : Σαν το παλιό καλό κρασί – Ο 8051

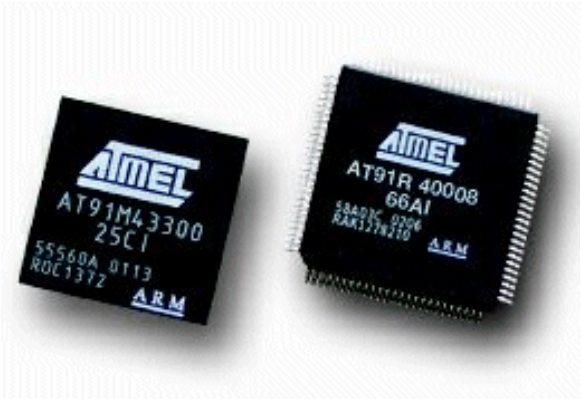
Ένας παραδοσιακός μικροεπεξεργαστής δεν θα σας επιτρέψει να κάνετε αυτό. Απαιτεί όλους αυτούς τους στόχους να αντιμετωπιστούν από άλλα τσιπ. Παραδείγματος χάριν, κάποιος αριθμός τσιπ μνήμης RAM πρέπει να προστεθεί. Το παρεχόμενο ποσό μνήμης έχει ανοχές στην παραδοσιακή προσέγγιση, αλλά τουλάχιστον μερικά εξωτερικά τσιπ μνήμης πρέπει να προστεθούν, και πρόσθετα απαιτούν ότι πολλές συνδέσεις πρέπει να γίνουν για να περάσουν τα δεδομένα από και προς αυτά.

Παραδείγματος χάριν, ένας χαρακτηριστικός μικροελεγκτής θα έχει χτισμένη στη γεννήτρια ρολογιών και ένα μικρό ποσό RAM ή ROM (ή EPROM ή EEPROM), που σημαίνει ότι για να τεθεί σε λειτουργία, όλα και όλα που απαιτούνται είναι κάποιο λογισμικό ελέγχου και ένα κρύσταλλο.



Εικόνα 2.12 : Ο PIC της MICROCHIP

Οι μικροελεγκτές επίσης συνήθως θα έχουν ποικίλες συσκευές εισόδου / εξόδου, όπως οι αναλογικό σε ψηφιακό μετατροπείς, τα χρονόμετρα, οι UARTs ή ειδικευμένα σειριακά interface επικοινωνιών όπως το I²C, το SPI και το Controller Area Network (CAN). Συχνά αυτές οι ενσωματωμένες συσκευές μπορούν να ελεγχθούν από εξειδικευμένες οδηγίες επεξεργαστών. Μερικοί σύγχρονοι μικροελεγκτές περιλαμβάνουν μια ενσωματωμένη υψηλού επιπέδου γλώσσα προγραμματισμού με την BASIC είναι αρκετά κοινή σε αυτό.



Εικόνα 2.13 : Ο AT91 της ATMEL

Οι μικροελεγκτές εμπορεύονται ταχύτητα και ευελιξία στον σχεδιασμό εξοπλισμού με το χαμηλότερο κόστος. Τέλος, πρέπει να αναφερθεί ότι μερικές αρχιτεκτονικές μικροελεγκτών είναι διαθέσιμες από πολλούς διαφορετικούς προμηθευτές σε τόσες πολλές ποικιλίες και θα μπορούσαν εύκολα να ανήκουν σε μια δική τους κατηγορία. Αρχηγός μεταξύ αυτών είναι ο 8051 και τα παράγωγα του Z80.

2.3 –Ο μικροελεγκτής AVR

Στην κατασκευή του MP3 αποκωδικοποιητή χρησιμοποιήθηκε ο μικροελεγκτής AVR της ATMEL, και ειδικότερα ο ATmega32, λόγω των πολλών δυνατοτήτων του αλλά και της χαμηλής τιμής του.

Εικόνα 2.14 : Ο ATmega32 της ATMEL σε DIP-40 πακετάρισμα

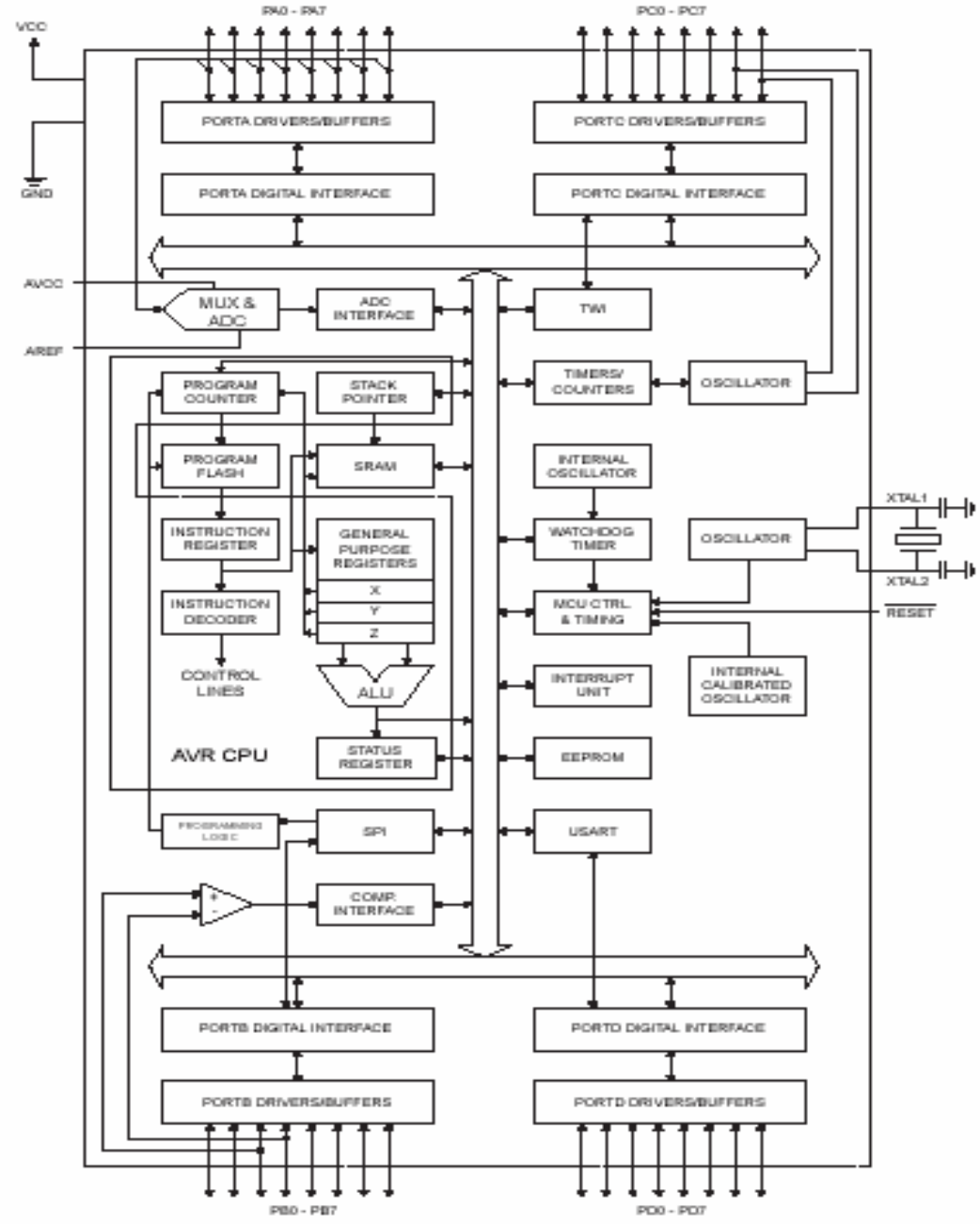
2.3.1 –Χαρακτηριστικά

Ειδικότερα τα χαρακτηριστικά που προσφέρει και τον κάνουν ιδιαίτερα ελκυστικό για τον σχεδιασμό ενός embedded συστήματος σαν αυτό που κατασκευάστηκε είναι τα παρακάτω:

- Μεγάλη απόδοση, μικρή κατανάλωση
- Αναβαθμισμένη RISC αρχιτεκτονική
  - 131 πολύ ισχυρές εντολές – οι περισσότερες απαιτούν μόνο ένα κύκλο ρολογιού για την εκτέλεση τους
  - 32 καταχωρητές μεγέθους 8-bit γενικής χρήσης
  - Μέχρι 16 MIPS στα 16MHz
  - On chip πολλαπλασιαστής 2 κύκλων
- Non volatile μνήμη προγράμματος και δεδομένων
  - 32 KB Αυτοπρογραμματιζόμενη μνήμη flash
  - 1 KB EEPROM μνήμη
  - 2 KB εσωτερική SRAM
- JTAG διεπαφή για εύκολο προγραμματισμό και debugging
- Πλήθος περιφερειακών
  - 2 timers 8-bit με ξεχωριστά ρολόγια
  - 1 timer 16-bit με ξεχωριστό ρολόι
  - Αληθινού χρόνου μετρητής με εξωτερικό κρύσταλλο
  - 4 κανάλια PWM
  - 8 κανάλια 10-bit ADC
  - TWI
  - Master/Slave SPI λειτουργία
  - Σειριακή θύρα με δυνατότητες σύγχρονης και ασύγχρονης λειτουργίας
  - Watchdog timer με ξεχωριστό κρύσταλλο
  - On-chip αναλογικό συγκριτή
- Ειδικά χαρακτηριστικά
  - Reset αυτόματα με την τροφοδότηση

- Εσωτερικό ρολόι
- Εσωτερικές και εξωτερικές διακοπές
- 6 sleep modes για εξοικονόμηση ενέργειας

- Ταχύτητα έως 16MHz
- Μικρή κατανάλωση



Εικόνα 2.15 : Μπλοκ διάγραμμα του ATmega32 (RISC)

2.3.2 – Γενική περιγραφή

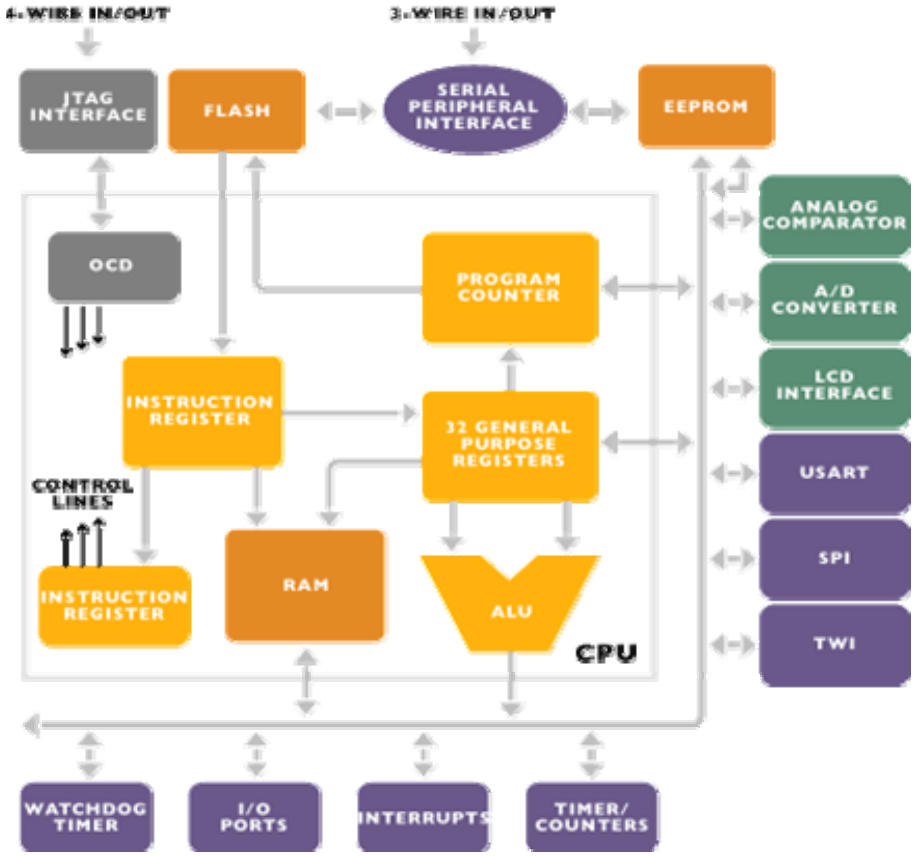
Ο ATmega32 είναι ένας χαμηλής κατανάλωσης CMOS 8-bit επεξεργαστής βασισμένος πάνω στην ενισχυμένη RISC αρχιτεκτονική των AVR. Εκτελώντας τις δυναμικές του εντολές σε ένα κύκλο μηχανής επιτρέπει στον σχεδιαστή να βελτιστοποιήσει την κατανάλωση σε σχέση με την επεξεργαστική ισχύ.

Ο σχεδιασμός του ATmega32 προβλέπει ένα πλούσιο σετ εντολών συνδεδεμένο με 32 καταχωρητές γενικής χρήση. Όλοι οι καταχωρητές συνδέονται άμεσα με την αριθμητική

λογική μονάδα (ALU) επιτρέποντας τις εντολές μεταξύ καταχωρητών να εκτελεστούν σε ένα κύκλο μηχανής πετυχαίνοντας έτσι μέχρι και δέκα φορές μεγαλύτερο throughput από τις CISC αρχιτεκτονικές. Επιπλέον ο ATmega32 έχει ένα πλήθος περιφερειακών έτσι ώστε να έχει την δυνατότητα να ελέγξει ένα ολόκληρο σύστημα με περιφερειακά μόνος του.

2.3.3 – Γενική περιγραφή αρχιτεκτονικής

Με σκοπό να μεγιστοποιήσει την απόδοση και τον παραλληλισμό στην εκτέλεση των εντολών ο AVR χρησιμοποιεί μία ειδικά σχεδιασμένη παραλλαγή της αρχιτεκτονικής Harvard – με ξεχωριστή μνήμη και buses για το πρόγραμμα και τα δεδομένα. Παρακάτω φαίνεται μία εικόνα της χρησιμοποιούμενης από τον AVR αρχιτεκτονικής.



Εικόνα 2.16 : Αρχιτεκτονική του AVR

Ο AVR χρησιμοποιεί pipeline ενός σταδίου. Ενώ μία εντολή εκτελείται η επόμενη ανακαλείται από την μνήμη προγράμματος. Αυτό οδηγεί στην εκτέλεση των εντολών σε ένα κύκλο μηχανής (οι περισσότερες από αυτές). Η μνήμη του προγράμματος είναι In-System Reprogrammable Flash.

Το Register File περιέχει 32 x 8 bit γενικής χρήσης καταχωρητές με χρόνο πρόσβασης ένα κύκλο ρολογιού. Οι καταχωρητές αυτοί είναι άμεσα συνδεδεμένοι με την αριθμητική λογική μονάδα επιτρέποντας την λειτουργία της ALU σε μία μονάδα χρόνου (ένα κύκλο ρολογιού).

Έξι από τους 32 καταχωρητές μπορούν να χρησιμοποιηθούν σαν 3 16–bit έμμεσοι καταχωρητές-δείκτες της μνήμης δεδομένων επιτρέποντας έτσι αποδοτικούς υπολογισμούς διευθύνσεων μνήμης. Ένας από αυτούς μπορεί να χρησιμοποιηθεί σαν δείκτης σε look-up πίνακες στην μνήμη του προγράμματος. Αυτοί οι καταχωρητές είναι οι X- , Y- , Z-.

Η αριθμητική λογική μονάδα επιτρέπει αριθμητικές και λογικές πράξεις μεταξύ δύο καταχωρητών ή μεταξύ ενός καταχωρητή και μίας σταθεράς. Ακόμα λειτουργίες ενός μόνο καταχωρητή εκτελούνται από τη ALU. Έπειτα από μία εκτέλεση λειτουργίας από την αριθμητική λογική μονάδα ο status-καταχωρητής ανανεώνεται για να απεικονίζει τα αποτελέσματα της λειτουργίας.

Η ροή του προγράμματος διευκολύνεται από με και χωρίς προϋποθέσεις εντολές άλματος και κλήσης που μπορούν άμεσα να μεταφέρουν την εκτέλεση του προγράμματος σε οποιοδήποτε σημείο της μνήμης προγράμματος. Οι περισσότερες εντολές του AVR έχουν μορφή 16-bit. Κάθε διεύθυνση στην μνήμη του προγράμματος περιέχει μία εντολή 16 ή 32 bit.

Η μνήμη προγράμματος είναι χωρισμένη σε δύο μέρη: την μνήμη εκκίνησης και την μνήμη για τις εφαρμογές. Όλες οι μνήμες στον AVR είναι γραμμικές.

Κατά την διάρκεια διακοπών και υπορουτίνων η διεύθυνση επιστροφής αποθηκεύεται στην στοίβα. Η στοίβα βρίσκεται στην SRAM και συνεπώς το μέγεθος της εξαρτάται μόνο από την μνήμη SRAM που διαθέτει ο κάθε επεξεργαστής και την χρησιμοποίηση της για αποθήκευση μεταβλητών. Ο δείκτης της στοίβας πρέπει να αρχικοποιείται κάθε φορά που αρχίζει ένα πρόγραμμα. Ο δείκτης της στοίβας είναι γενικά προσβάσιμος στην περιοχή I/O του μικροελεγκτή. Η μνήμη SRAM είναι προσβάσιμη μέσω πέντε τρόπων διευθυνσιοδότησης στην αρχιτεκτονική του AVR.

Ακόμα αξιοσημείωτα στην αρχιτεκτονική του AVR είναι το ευέλικτο σύστημα για την διαχείριση των διακοπών και οι 64 διευθύνσεις για περιφερειακές συσκευές.

2.3.4 – Δομή της μονάδας επεξεργασίας του AVR

Παρακάτω αναλύεται η κεντρική μονάδα επεξεργασίας του μικροελεγκτή AVR.

2.3.4.1 – Αριθμητική Λογική Μονάδα (ALU)

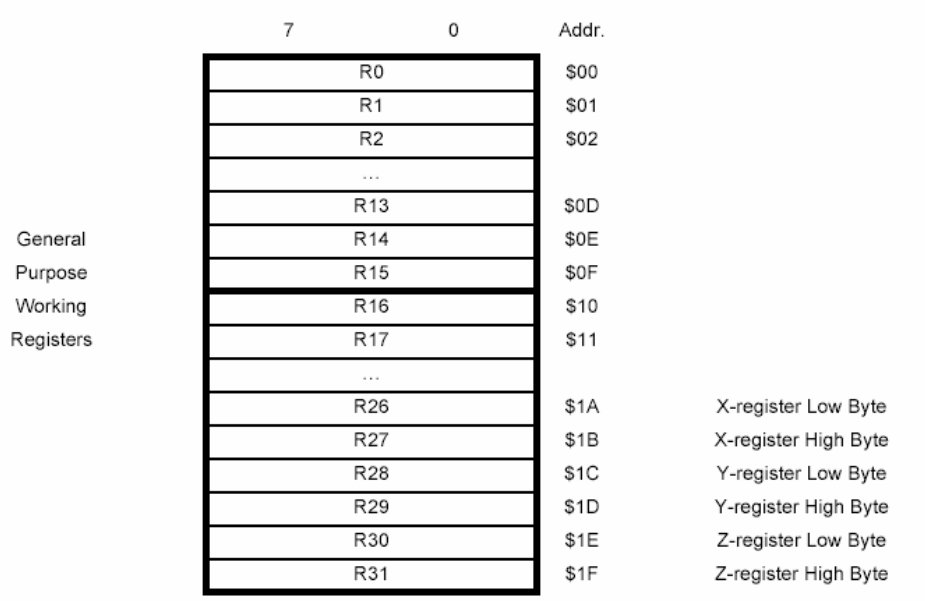
Η υψηλής απόδοσης ALU λειτουργεί σε απευθείας σύνδεση με τους 32 γενικής χρήσης καταχωρητές. Μέσα σ' ένα κύκλο ρολογιού εκτελούνται οι αριθμητικές πράξεις μεταξύ δύο καταχωρητών ή μεταξύ ενός καταχωρητή και ενός ορίσματος από την μνήμη. Οι λειτουργίες της ALU διακρίνονται σε αριθμητικές, λογικές και λειτουργίες σε bit επίπεδο. Ακόμα υποστηρίζεται δυναμική χρήση πολλαπλασιασμού τόσο σε αριθμούς χωρίς πρόσημο όσο και σε προσημασμένους αριθμούς.

2.3.4.2 – Οι καταχωρητές γενικής χρήσης του AVR

Οι καταχωρητές του AVR είναι κατασκευασμένοι έτσι ώστε να υποστηρίζουν το ενισχυμένο RISC ρεπερτόριο εντολών. Έτσι υποστηρίζονται οι εξής λειτουργίες από τους καταχωρητές:

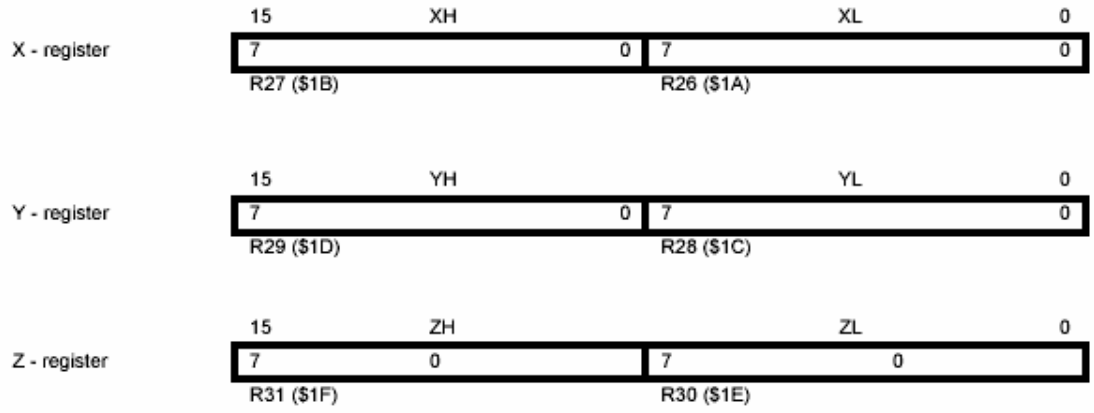
1. Ένα 8-bit όρισμα έξοδος και ένα 8-bit αποτέλεσμα είσοδος
2. Δύο 8-bit ορίσματα έξοδος και ένα 8-bit αποτέλεσμα είσοδος
3. Δύο 8-bit ορίσματα έξοδος και ένα 16-bit αποτέλεσμα είσοδος
4. Ένα 16-bit όρισμα έξοδος και ένα 16-bit αποτέλεσμα είσοδος

Παρακάτω φαίνονται οι καταχωρητές του AVR:



Εικόνα 2.17 : Καταχωρητές γενικής χρήσης του AVR

Οι περισσότερες από τις εντολές που λειτουργούν στο αρχείο καταχωρητών έχουν άμεση πρόσβαση σε όλους τους καταχωρητές, και οι περισσότερες από αυτές εκτελούνται σε ένα κύκλο. Όπως φαίνεται στο σχήμα 4, σε κάθε καταχωρητή ορίζεται επίσης μια διεύθυνση μνήμης δεδομένων, χαρτογραφώντας τους άμεσα στις πρώτες 32 θέσεις του διαστήματος δεδομένων. Αν και όχι φυσικά ως θέσεις SRAM, αυτή η οργάνωση μνήμης παρέχει τη μεγάλη ευελιξία στην πρόσβαση των καταχωρητών, όπως το X -, Y -, και οι καταχωρητές Z-δεικτών μπορούν να τεθούν ως δείκτης για να συντάξουν ευρετήριο σε οποιοδήποτε καταχωρητή στο αρχείο.



Εικόνα 2.18 : Οι διπλοί καταχωρητές X- , Y- , Z- του AVR

2.3.4.3 – Οι δείκτης στοίβας του AVR

Η στοίβα χρησιμοποιείται κυρίως για την αποθήκευση των προσωρινών στοιχείων, για την αποθήκευση των τοπικών μεταβλητών και για την αποθήκευση των διευθύνσεων επιστροφής μετά από διακοπές και κλήσεις υπορουτινών. Ο καταχωρητής-δείκτης στοίβας δείχνει πάντα την κορυφή της στοίβας. Σημειώστε ότι η στοίβα εφαρμόζεται αυξανόμενη από τις υψηλότερες θέσεις μνήμης στις χαμηλότερες θέσεις μνήμης. Αυτό υπονοεί ότι μια εντολή push της στοίβας μειώνει το δείκτη στοίβας.

Ο δείκτης στοίβας δείχνει την περιοχή στοίβας της SRAM όπου οι στοίβες υπορουτινών και διακοπών βρίσκονται. Αυτό το διάστημα στοίβας στην SRAM πρέπει να

καθοριστεί από το πρόγραμμα προτού να εκτελεσθούν οποιεσδήποτε κλήσεις υπορουτινών ή επιτρεπόμενων διακοπών. Ο δείκτης στοίβας πρέπει να τεθεί ως δείκτης και να δείξει επάνω από $60. Ο δείκτης στοίβας μειώνεται κατά ένα όταν ένα δεδομένο τοποθετείται στην στοίβα με την push εντολή, και μειώνεται κατά δύο όταν γίνεται push η διεύθυνση επιστροφής στη στοίβα με την κλήση υπορουτινών ή διακοπών. Αντίστοιχες αυξήσεις γίνονται όταν εκτελούνται pop εντολές.

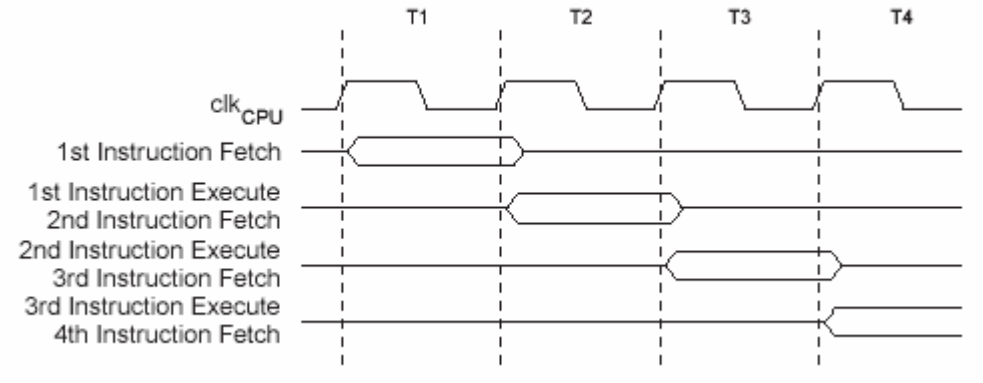
Ο δείκτης στοίβας AVR κατασκευάζεται ως δύο οκτάμπιτοι καταχωρητές στο διάστημα I/O. Ο αριθμός bits που χρησιμοποιείται είναι στην πραγματικότητα εφαρμογή εξαρτώμενη. Σημειώστε ότι τα δεδομένα σε μερικές εφαρμογές της αρχιτεκτονικής AVR είναι τόσο μικρά που μόνο ο SPL απαιτείται. Σε αυτήν την περίπτωση, ο καταχωρητής SPH δεν θα χρησιμοποιηθεί.

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

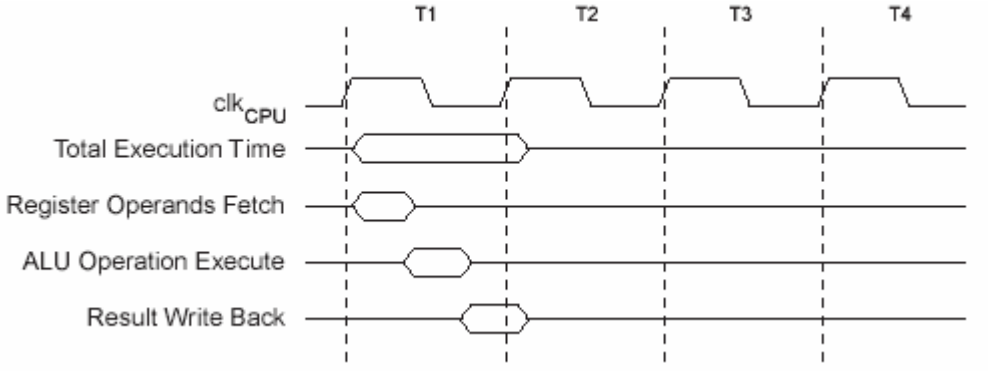
Εικόνα 2.19 : Ο δείκτης στοίβας SP του AVR

2.3.4.4 – Χρόνοι εκτέλεσης εντολών

Η αναβαθμισμένη Harvard αρχιτεκτονική που διαθέτει ο AVR οδηγεί σε ένα μεγάλο παραλληλισμό εκτέλεσης εντολών κάτι που τον κάνει να φθάνει σχεδόν σε απόδοση το 1 MIPS ανά MHz ρολογιού. Στα παρακάτω διαγράμματα φαίνεται η εκτέλεση συνεχόμενων εντολών από την ALU καθώς και τα στάδια εκτέλεσης μίας εντολής.



Εικόνα 2.20 : Εκτέλεση εντολών από την ALU



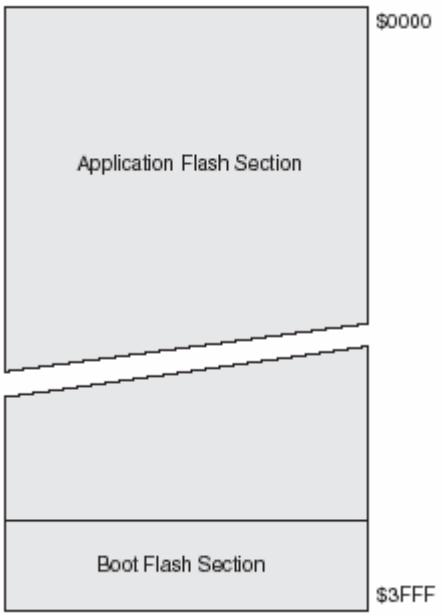
Εικόνα 2.21 : Στάδια εκτέλεσης εντολής από την ALU

2.3.5 – Μνήμες του AVR

Τα τμήματα μνήμης του AVR είναι βασισμένα στο μοντέλο αρχιτεκτονικής Harvard στο οποίο τα σημαντικά τμήματα της μνήμης είναι διαχωρισμένα για να επιτυχαίνεται ταχύτερη πρόσβαση σε αυτά και αυξημένη χωρητικότητα. Η CPU έχει ξεχωριστό interface πρόσβασης στην μνήμη προγράμματος FLASH, την μνήμη δεδομένων και στην EEPROM (αν υπάρχει).

2.3.5.1 – Η μνήμη προγράμματος FLASH

Η μνήμη προγράμματος FLASH είναι ένα block μνήμης FLASH η οποία αρχίζει από την διεύθυνση $0000 και το μέγεθος της εξαρτάται από το μοντέλο του AVR. (Ειδικότερα το όνομα του AVR προδίδει το μέγεθος της μνήμης. Ο ATmega32 παραδείγματος χάριν που χρησιμοποιήθηκε έχει μνήμη FLASH 32KB. Παρόμοια ο ATmega128 έχει 128KB FLASH). Η μνήμη FLASH είναι non-volatile (δηλαδή διατηρεί τα δεδομένα της μετά την απομάκρυνση της τροφοδοσίας) και έχει διάρκεια ζωής 10,000 κύκλους εγγραφής διαγραφής. Χρησιμοποιείται για την αποθήκευση του κώδικα του προγράμματος και σταθερών. Η διευθυσιοδότηση της μνήμης είναι της τάξης των 16 bits. Παρακάτω φαίνεται μία εικόνα της μνήμης FLASH



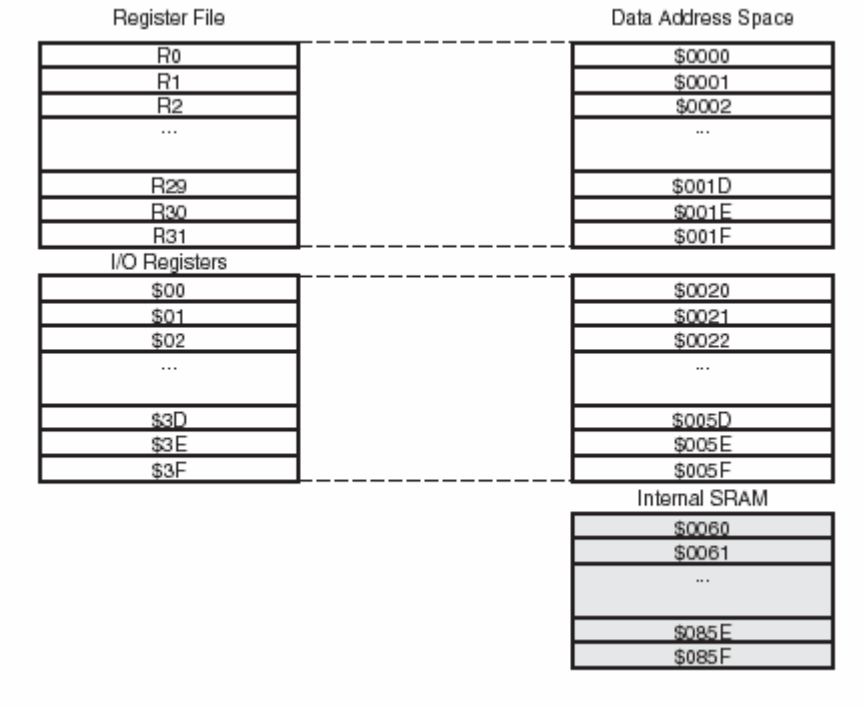
Εικόνα 2.22 : Μνήμη προγράμματος FLASH

Παρόλο που η μνήμη είναι επανεγγράψιμη δεν μπορεί να εγγραφεί μέσω ενός εκτελέσιμου προγράμματος, πρέπει να προγραμματιστεί από εξωτερικά μέσα. Συνεπώς είναι μία read-only μνήμη και εκεί μπορούν να αποθηκευτούν μόνο σταθερές μαζί με τον εκτελέσιμο κώδικα. Οι σταθερές όταν αποθηκεύονται στην μνήμη FLASH αυτόματα προβιβάζονται σε ακεραίους λόγω του μεγέθους της μονάδας αποθήκευσης.

2.3.5.2 – Η μνήμη δεδομένων SRAM

Η μνήμη δεδομένων του AVR περιέχει τρεις περιοχές μνήμης ανάγνωσης / εγγραφής. Το χαμηλότερο σε διεύθυνση τμήμα της μνήμης περιέχει τους 32 γενικής χρήσης καταχωρητές ακολουθούμενο από τους 64 καταχωρητές εισόδου εξόδου (το νούμερο εξαρτάται από το μοντέλο του επεξεργαστή) και την εσωτερική SRAM. Οι γενικής χρήσης καταχωρητές χρησιμοποιούνται για την αποθήκευση global μεταβλητών καθώς και άλλων προσωρινών δεδομένων κατά την εκτέλεση του προγράμματος. Οι 64 καταχωρητές εισόδου

– εξόδου χρησιμοποιούνται για την επικοινωνία με τις συσκευές εισόδου – εξόδου και των άλλων περιφερειακών.

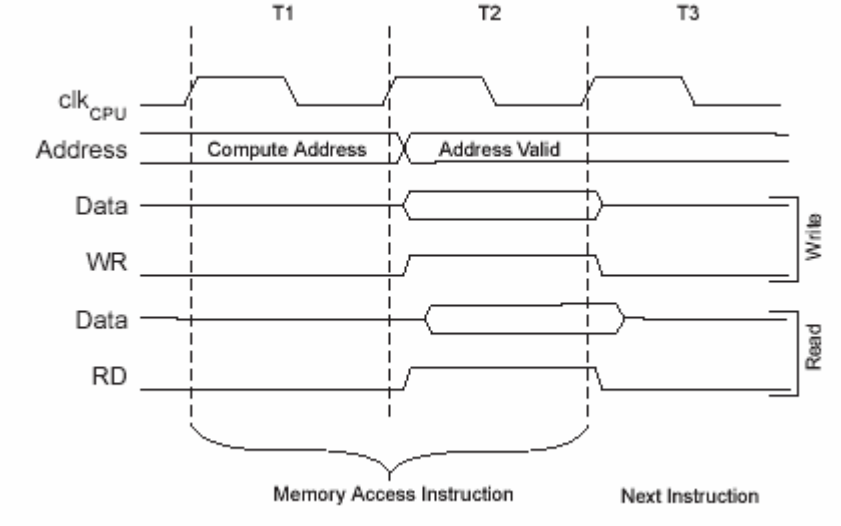


Εικόνα 2.23 : Μνήμη δεδομένων SRAM

Η SRAM μνήμη (το υπόλοιπο τμήμα) χρησιμοποιείται για την αποθήκευση μεταβλητών που δεν χωράνε στους καταχωρητές και για τη δημιουργία της στοίβας.

Αυτό το κομμάτι δεν έχει ειδικές διαιρέσεις. Τα δεδομένα συνήθως αποθηκεύονται αρχίζοντας από το πρώτο κομμάτι της μνήμης ενώ η στοίβα αποθηκεύεται στο υψηλότερο κομμάτι της μνήμης. Ο ATmega32 έχει 2048 bytes SRAM.

Παρακάτω φαίνεται ο χρόνος προσπέλασης της SRAM.



Εικόνα 2.24 : Χρόνοι προσπέλασης SRAM

2.3.5.3 - Η μνήμη EEPROM

Η μνήμη EEPROM είναι μία μνήμη που χρησιμοποιείται για την εγγραφή / ανάγνωση δεδομένων τα οποία πρέπει να διατηρούνται όταν αποκόπτεται η τροφοδοσία. Για αυτό το λόγο είναι μία non-volatile μνήμη. Συνήθως αρχίζει από την διεύθυνση 0x000

και πηγαίνει μέχρι το μέγιστο κάθε μικροϋπολογιστή (στην περίπτωση του ATmega32 τα 1024 bytes). Επειδή έχει αργό χρόνο προσπέλασης και περιορισμένο μέγιστο αριθμό κύκλων εγγραφής / ανάγνωσης χρησιμοποιείται κυρίως για αποθήκευση μεταβλητών που πρέπει να διατηρηθούν οι τιμές του σε μία διακοπή της τροφοδοσίας.

2.3.6 – Διακοπές του AVR – Reset

Οι διακοπές είναι ουσιαστικά κλήσεις συναρτήσεων από το Hardware. Όπως προδίδει και το όνομα τους οι διακοπές διακόπτουν την ροή του κυρίου προγράμματος και τον κάνουν να συνεχίσει την εκτέλεση του από ένα σημείο που είναι η ρουτίνα εξυπηρέτησης της διακοπής. Οι διακοπές είναι χρήσιμες για αυτές τις περιπτώσεις που ο επεξεργαστής πρέπει να ανταποκριθεί αμέσως σ' ένα γεγονός ή σε περιπτώσεις που αντί να περιμένει ένα αργό γεγονός να συμβεί αφήνει την διακοπή να τον ειδοποιήσει.

Πίνακας 2.1 : Διανύσματα διακοπών και Reset

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Ο AVR παρέχει διάφορες διακοπών που φαίνονται στον παραπάνω πίνακα. Όλες αυτές μαζί με το ξεχωριστό διάνυσμα διακοπής Reset έχουν ένα ξεχωριστό διάνυσμα στην μνήμη προγράμματος. Όλες οι διακοπές έχουν ξεχωριστά bits ενεργοποίησης τα οποία για να ενεργοποιήσουν μία διακοπή πρέπει να γραφούν με λογικό 1 και ταυτόχρονα να είναι ενεργοποιημένες οι διακοπές στον Status καταχωρητή.

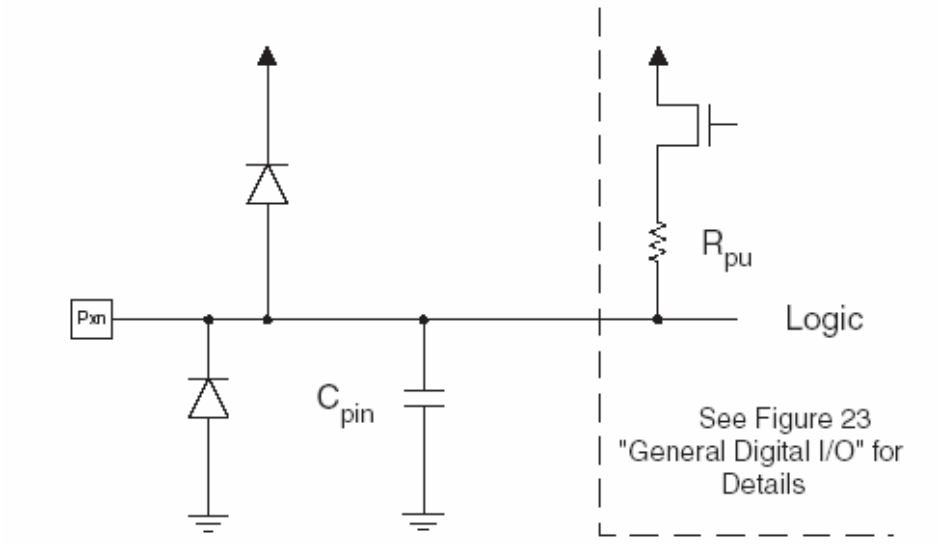
Στο χαμηλότερο κομμάτι του προγράμματος μνήμης έχει by default ανατεθεί ο χώρος στον οποίο βρίσκονται τα διανύσματα των διακοπών. Η σειρά αυτή καθορίζει και την προτεραιότητα των διακοπών. Όσο πιο μικρή είναι η διεύθυνση της διακοπής τόσο πιο μεγάλη είναι η προτεραιότητα της. Την μεγαλύτερη προτεραιότητα έχει η διακοπή Reset και αμέσως επόμενη είναι η εξωτερική διακοπή Int0. Αυτό δεν σημαίνει ότι μία διακοπή μπορεί να σταματήσει την ρουτίνα εξυπηρέτησης μίας διακοπής (καθώς όταν συμβαίνει μία διακοπή απενεργοποιούνται οι διακοπές) με μικρότερη προτεραιότητα καθώς αυτό δεν μπορεί να γίνει αλλά ότι αν ταυτόχρονα περιμένουν να γίνουν δύο διακοπές πρώτη θα εκτελεστεί η διακοπή με την μεγαλύτερη προτεραιότητα.

Βασικά υπάρχουν δύο είδη διακοπών. Το πρώτο είδος συμβαίνει από γεγονότα που θέτουν την σημαία της διακοπής. Τότε ο επεξεργαστής πηγαίνει στην διεύθυνση που δείχνει το διάνυσμα διακοπής για να εκτελέσει την ρουτίνα εξυπηρέτησης της διακοπής. Έπειτα επιστρέφει στην εκτέλεση του κυρίως προγράμματος και καθαρίζει την σημαία της διακοπής. Το δεύτερο είδος προέρχεται από γεγονότα τα οποία προκαλούν διακοπές όσο είναι ενεργά και δεν έχουν απαραίτητα σημαίες διακοπής.

Ο χρόνος ανταπόκρισης σε μία διακοπή είναι τουλάχιστον τέσσερις κύκλοι ρολογιού. Μετά από τέσσερις κύκλους ρολογιού η ρουτίνα εξυπηρέτησης της διακοπής εκτελείται. Κατά την διάρκεια του τέταρτου κύκλου ο PC αποθηκεύεται στην στοίβα Έπειτα εκτελείται ένα jump για την ρουτίνα εξυπηρέτησης της διακοπής που παίρνει τρεις κύκλους. Όταν επιστρέφει το πρόγραμμα από μία ISR ο PC ανακτάται από την στοίβα, ο SP αυξάνεται κατά δύο και ενεργοποιούνται οι διακοπές στον Status καταχωρητή. Αυτό παίρνει τέσσερις κύκλους ρολογιού.

2.3.7 – Παράλληλες θύρες εισόδου / εξόδου του AVR

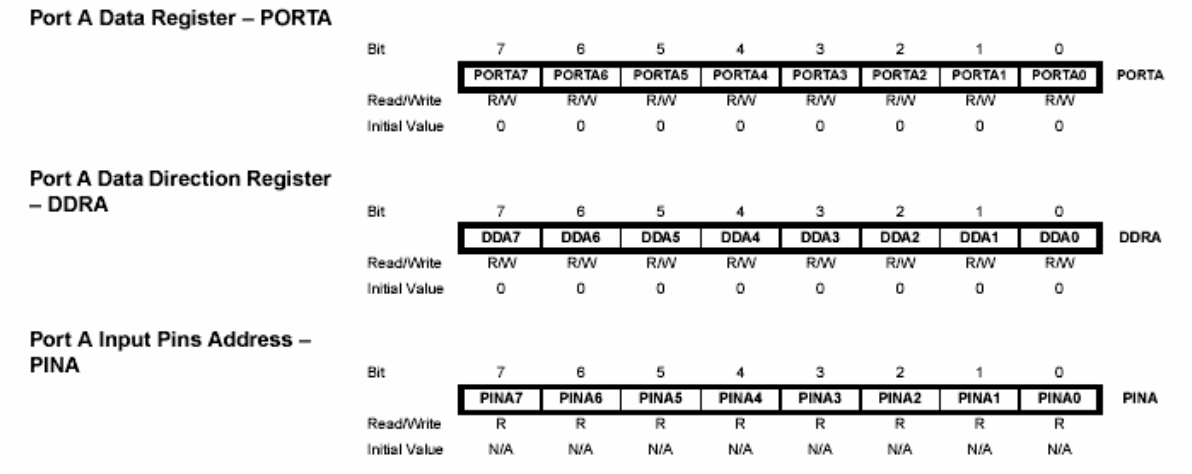
Όλες οι παράλληλες θύρες εισόδου – εξόδου του AVR έχουν αυτονομία όταν χρησιμοποιούνται σαν γενικές θύρες εισόδου – εξόδου. Αυτό σημαίνει ότι μπορείς να αλλάξεις την κατάσταση ενός μόνο bit στην πόρτα χωρίς να πειράξεις κανένα άλλο. Το ίδιο συμβαίνει και με την διεύθυνση της πόρτας η οποία μπορεί να αλλάξει από είσοδος σε έξοδος.



Εικόνα 2.25 : Διάγραμμα πόρτας εισόδου – εξόδου του AVR

Κάθε πόρτα του AVR έχει για τον έλεγχο της 3 καταχωρητές, τους PORTx, PINx και DDRx. Ανάλογα με την τιμή του DDRx η πόρτα μπορεί να χρησιμοποιηθεί σαν είσοδος

ή έξοδος. Ο PORTx χρησιμοποιείται για την θέση της τιμής της πόρτας και ο PINx για την ανάγνωση της τιμής της πόρτας.



Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Εικόνα 2.26 : Καταχωρητές πόρτας εισόδου – εξόδου του AVR

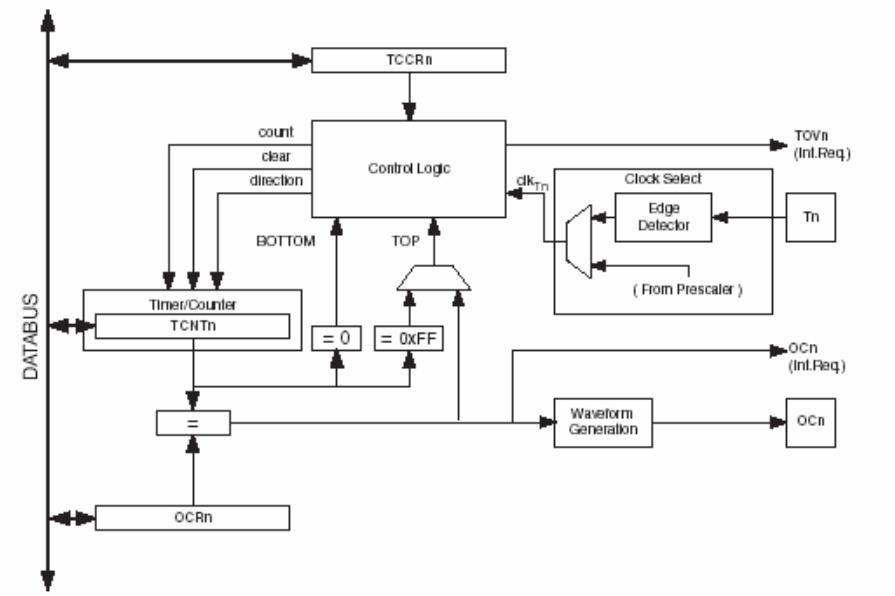
Εκτός από πόρτες γενικής χρήσης οι πόρτες του AVR μπορούν να ενεργοποιηθούν για να επιτελέσουν λειτουργίες εισόδου εξόδου άλλων περιφερειακών του μικροελεγκτή.

2.3.8 – Timers του AVR

Ο ATmega32 έχει τρεις χρονιστές / μετρητές για την επιτέλεση διάφορων διεργασιών. Οι timers είναι πιθανότατα το πιο ευρέως χρησιμοποιημένο περιφερειακό σε ένα μικροελεγκτή. Αυτό συμβαίνει λόγω των πολλών εφαρμογών που μπορούν να επιτελέσουν. Μέσω ενός timer μπορούν να μετρηθούν περίοδοι χρόνου, να καθοριστεί το εύρος παλμών, να μετρηθεί η ταχύτητα και η συχνότητα ή να δημιουργηθούν διάφορων ειδών σήματα.

Παρόλο που υπάρχουν δύο διαχωρισμένες μορφές, χρονομέτρηση και μέτρηση, οι timers είναι ουσιαστικά απλοί δυαδικοί μετρητές προς τα πάνω. Όταν χρησιμοποιούνται σε timing mode, οι δυαδικοί μετρητές μετρούν περιόδους ρολογιού που εφαρμόζονται στην είσοδο τους ενώ στην timing mode μετρούν παλμούς στην είσοδο τους. Οι timers μπορούν να χρησιμοποιήσουν ως είσοδο είτε υποδιαιρέσεις του ρολογιού ου μικροελεγκτή είτε εξωτερική πηγή ρολογιού.

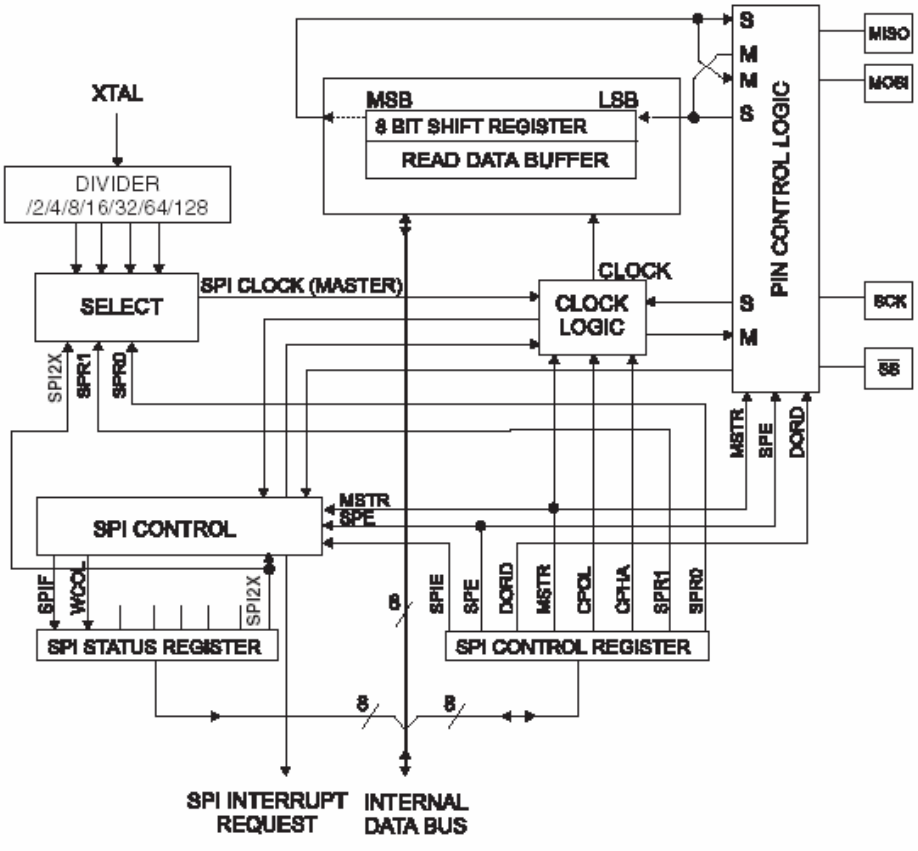
Ο ATmega32 έχει δύο 8-bit timers και έναν 16-bit timer. Εικόνα του timer 0 φαίνεται παρακάτω:



Εικόνα 2.27 : Μπλοκ διάγραμμα του timer 0

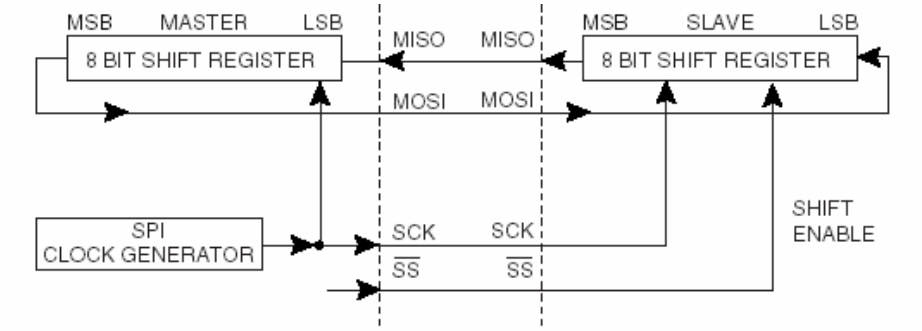
2.3.9 – SPI

Το SPI είναι ένα πρωτόκολλο σειριακής επικοινωνίας διαθέσιμο στον AVR . Είναι σύγχρονη σειριακή επικοινωνία διαδρόμου, που σημαίνει ότι τόσο ο αποστολέας τόσο και ο παραλήπτης πρέπει να χρησιμοποιούν το ίδιο ρολόι για να συγχρονίσουν την παραλαβή των bits. Το SPI αναπτύχθηκε για την γρήγορη και σχετικά πολύ υψηλής ταχύτητας επικοινωνία χρησιμοποιώντας το ελάχιστο των γραμμών.



Εικόνα 2.28 : Μπλοκ διάγραμμα του SPI

Η SPI επικοινωνία χρησιμοποιεί ένα master και ένα slave. Και οι δύο μεταδίδουν και λαμβάνουν δεδομένα ταυτόχρονα, αλλά ο master είναι υπεύθυνος για την δημιουργία του ρολογιού που είναι απαραίτητο για την μεταφορά των δεδομένων. Έτσι ο master έχει τον έλεγχο της ταχύτητας επικοινωνίας. Το παρακάτω σχηματικό διάγραμμα δείχνει πως επικοινωνούν δύο συσκευές μέσω του SPI. Ο master παρέχει το ρολόι και 8 bits δεδομένων τα οποία εξέρχονται ένα-ένα από τη MOSI έξοδο. Καθώς αυτά τα δεδομένα παραλαμβάνονται ένα προς ένα από τον slave, 8-bits στέλνονται ένα προς ένα από τον slave στον master στην MISO γραμμή. Έτσι οι δύο τους ανταλλάσσουν δεδομένα σε μία χρονική περίοδο.



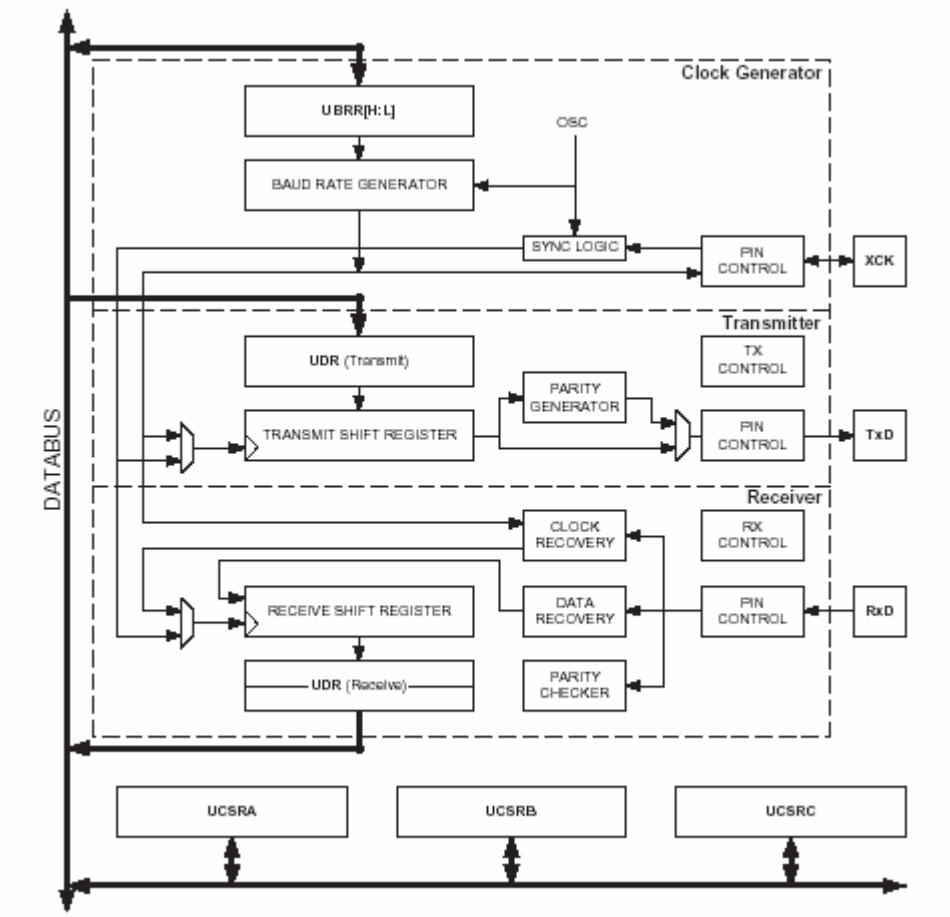
Εικόνα 2.29 : Επικοινωνία μέσω SPI

Πολλές συσκευές ταυτόχρονα μπορούν να συνδεθούν σε ένα διάδρομο SPI. Κάθε slave συσκευή απαιτεί ένα σήμα επιλογής το οποίο συνδέεται με τον μικροελεγκτή (ή κάποια άλλη συσκευή που είναι ο master στον διάδρομο) και γίνεται 0 όταν επιλέγεται η συσκευή.

2.3.10 – Σειριακή θύρα - USART

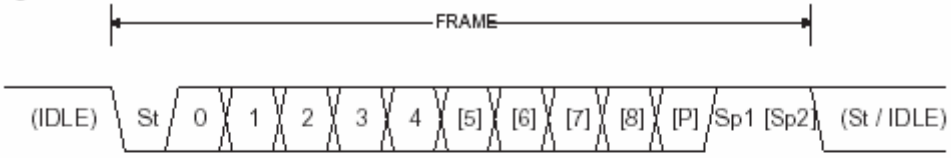
Η σειριακή επικοινωνία μέσω της USART είναι ουσιαστικά η αποστολή δεδομένων μέσω μίας γραμμής μόνο. Τα bits των δεδομένων ξεχωρίζουν από τον χρόνο που μεσολαβεί μεταξύ τους δίνοντας την δυνατότητα στον παραλήπτη να καταλάβει την αξία του κάθε bit.

Η USART λέγεται ασύγχρονη επικοινωνία λόγω του ότι δεν χρειάζεται μία γραμμή ρολογιού για να συγχρονιστούν τα bits. Η USART χρησιμοποιεί ένα start / stop bit για να ξεχωρίσει την αποστολή των δεδομένων.



Εικόνα 2.30 : Μπλοκ διάγραμμα της USART του ATmega32

Η επικοινωνία μέσω USART μένη ανενεργή στο λογικό 1 και πέφτει στο 0 για να σημειώσει την αρχή μίας νέας μεταφοράς (start bit). Ακολουθεί ένα πακέτο δεδομένων με μέγεθος από 5 έως 9 bits και έπειτα το stop bit. Η πτώση της γραμμής από 1 σε 0 ενεργοποιεί την μέτρηση χρόνου στον δέκτη ο οποίος δειγματοληπτεί την είσοδο του στο μέσο του ρολογιού για μέγιστη ακρίβεια.. Η USART έχει την δυνατότητα μεταφοράς δεδομένων σε διάφορα bitrates με μέγιστο για τον ATmega32 τα 115200 bps.



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

Εικόνα 2.31 : Frame δεδομένων

2.3.11 – TWI

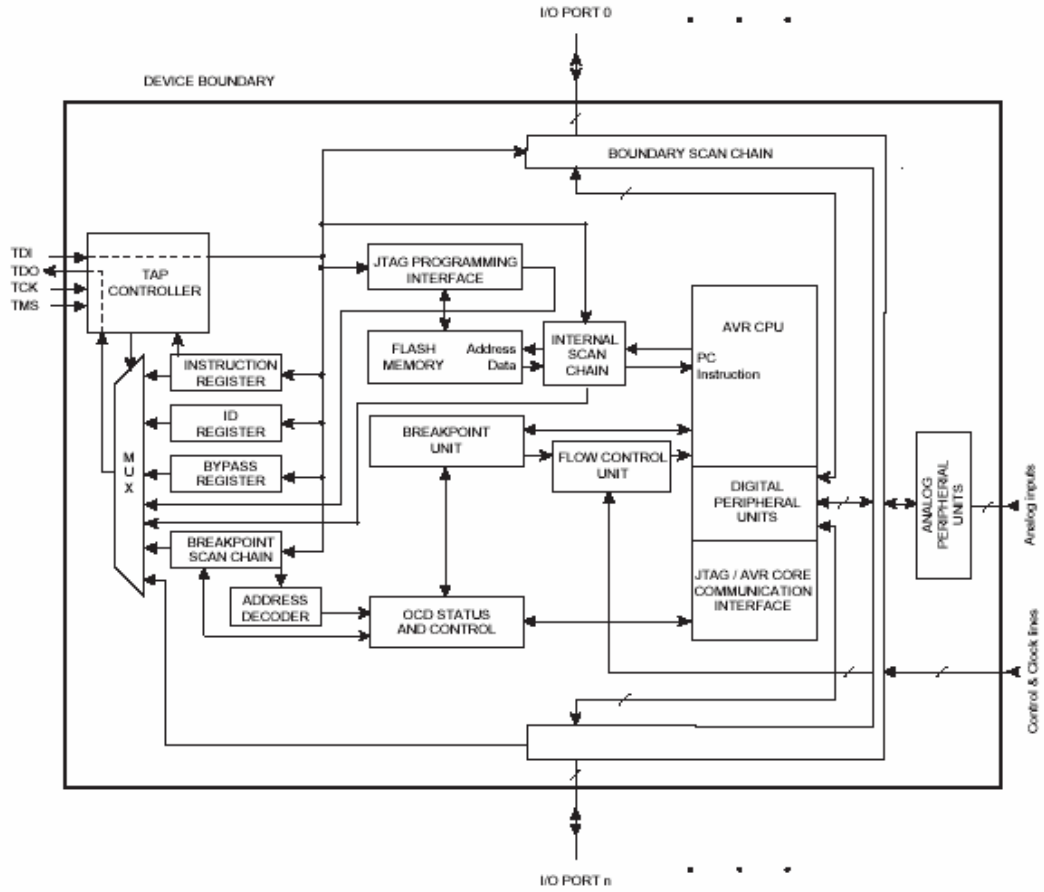
Ο ATmega32 υποστηρίζει ακόμα ένα interface σειριακής επικοινωνίας, το TWI το οποίο είναι πανομοιότυπο με το I2C αλλά λόγοι copyright απέτρεψαν στην χρησιμοποίηση του ονόματος αυτού. Το TWI interface είναι ιδανικό για την επικοινωνία του επεξεργαστή με μέχρι 128 I2C συσκευές πάνω στον ίδιο διάδρομο, με χρησιμοποίηση μόνο δύο γραμμών και απαίτηση μόνο μίας εξωτερικής pull-up αντίστασης. Λόγω της εκτενούς αναφοράς στο πανομοιότυπο I2C σε μετέπειτα κεφάλαιο θα παραλείψουμε εδώ την περαιτέρω αναφορά στο TWI.

2.3.12 – Υπόλοιπα περιφερειακά

Ο ATmega32 υποστηρίζει ένα πλήθος άλλων περιφερειακών. Θα γίνει μία σύντομη αναφορά στο καθένα καθώς δεν χρησιμοποιήθηκαν στην κατασκευή του mp3 αποκωδικοποιητή.

2.3.12.1 – JTAG

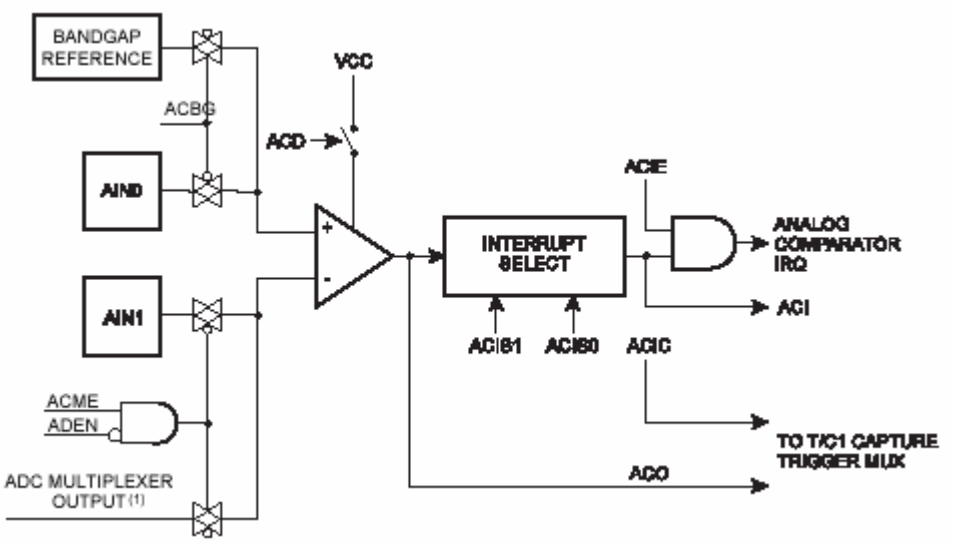
ΤΟ JTAG χρησιμοποιείται για on – system απασφαλμάτωση και προγραμματισμό των μικροελεγκτών. Ένα μπλοκ διάγραμμα του JTAG φαίνεται στο παρακάτω σχήμα.



Εικόνα 2.32 : Μπλοκ διάγραμμα του JTAG

2.3.12.2 – Αναλογικός συγκριτής

Ο αναλογικός συγκριτής συγκρίνει δύο αναλογικές εισόδους του μικροελεγκτή και θέτει σε λογικό0 ή 1 μία πόρτα εξόδου του μικροελεγκτή ανάλογα την τιμή τους.

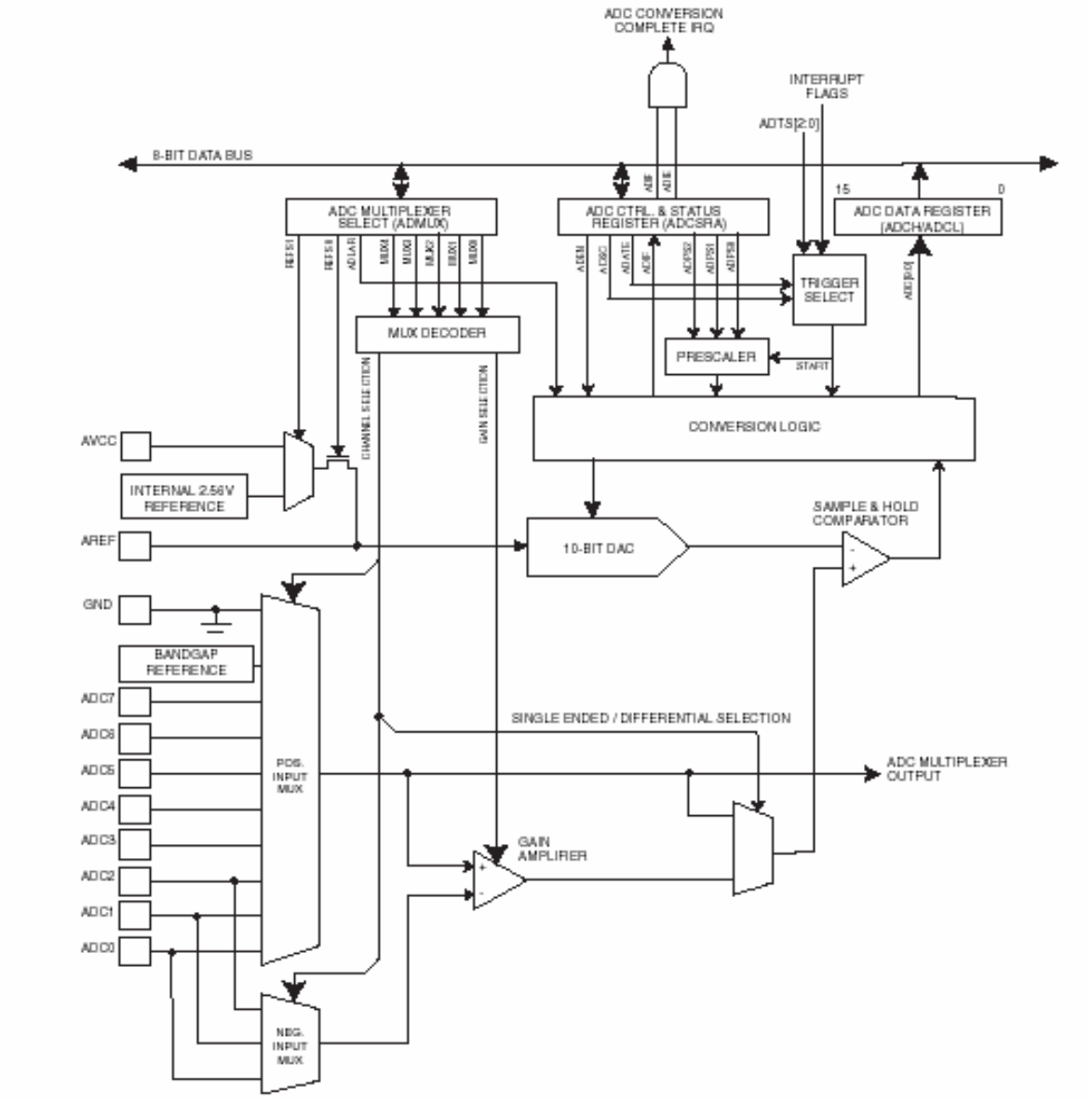


Εικόνα 2.33 : Αναλογικός συγκριτής

2.3.12.3 – Αναλογικό σε ψηφιακό μετατροπέας

Ο ATmega32 υποστηρίζει μια διαδοχική προσέγγιση ADC 10-bits. Ο ADC συνδέεται με έναν αναλογικό πολυπλέκτη 8-καναλιών που επιτρέπει την κατασκευή από 8 single-ended τάσεις. Οι single-ended εισαγωγές τάσης αναφέρονται στα 0V (gnd).

Η συσκευή υποστηρίζει επίσης 16 διαφορικούς συνδυασμούς εισαγωγής τάσης. Δύο από τις διαφορικές εισαγωγές (ADC1, ADC0 και ADC3, ADC2) είναι εξοπλισμένες με ένα προγραμματίσημο στάδιο κέρδους, που παρέχει τα βήματα ενίσχυσης 0 DB (1x), 20 DB (10x), ή 46 DB (200x) στη διαφορική τάση εισαγωγής πριν από τη μετατροπή A/D. Επτά διαφορικά αναλογικά κανάλια εισαγωγής μοιράζονται ένα κοινό αρνητικό τερματικό (ADC1), ενώ οποιαδήποτε άλλη εισαγωγή του ADC μπορεί να επιλεχτεί ως θετικό τερματικό εισαγωγής. Εάν 1x ή 10x κέρδος χρησιμοποιείται, οκτάμπιτη ανάλυση μπορεί να αναμένεται. Εάν 200x το κέρδος χρησιμοποιείται, τότε επτάμπιτη ανάλυση μπορεί να αναμένεται. Ακόμα υποστηρίζεται χρησιμοποίηση διαφορετικής από την γείωση τάση αναφοράς.



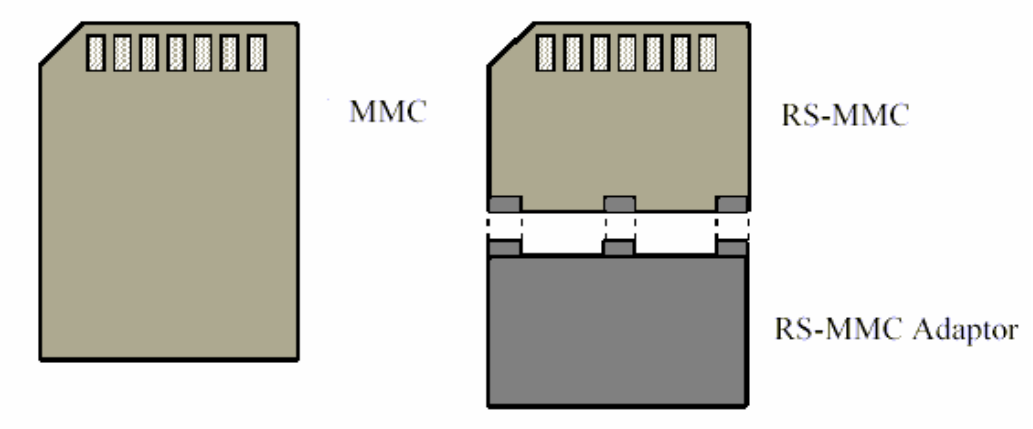
Εικόνα 2.34 : Αναλογικό σε ψηφιακό μετατροπέας

Κεφάλαιο 3

Τα περιφερειακά του συστήματος

3.1 - Κάρτα μνήμης (MultiMedia Card)

Το μέσο αποθήκευσης των αρχείων MP3 ,που χρησιμοποιήθηκε στην κατασκευή του αποκωδικοποιητή είναι μία multimedia κάρτα, η οποία θα αναφέρεται από εδώ και στο εξής ως MMC (MultiMedia Card). Ειδικότερα χρησιμοποιήθηκε μία RS-MMC (Reduced-Size MMC) της SanDisk Corporation ,η οποία προτιμήθηκε λόγω των πολλών πλεονεκτημάτων που προσφέρει στην σχεδίαση φορητών συστημάτων σε σχέση με άλλα μέσα αποθήκευσης και τα οποία αναφέρονται αναλυτικά παρακάτω.



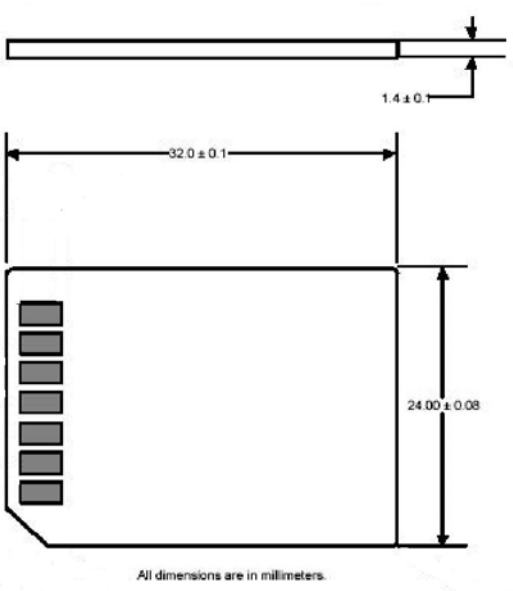
Εικόνα 3.1 : Πίσω όψη MMC και RS-MMC

Εικόνα 3.2 : RS-MMC της SanDisk χωρητικότητας 256MB

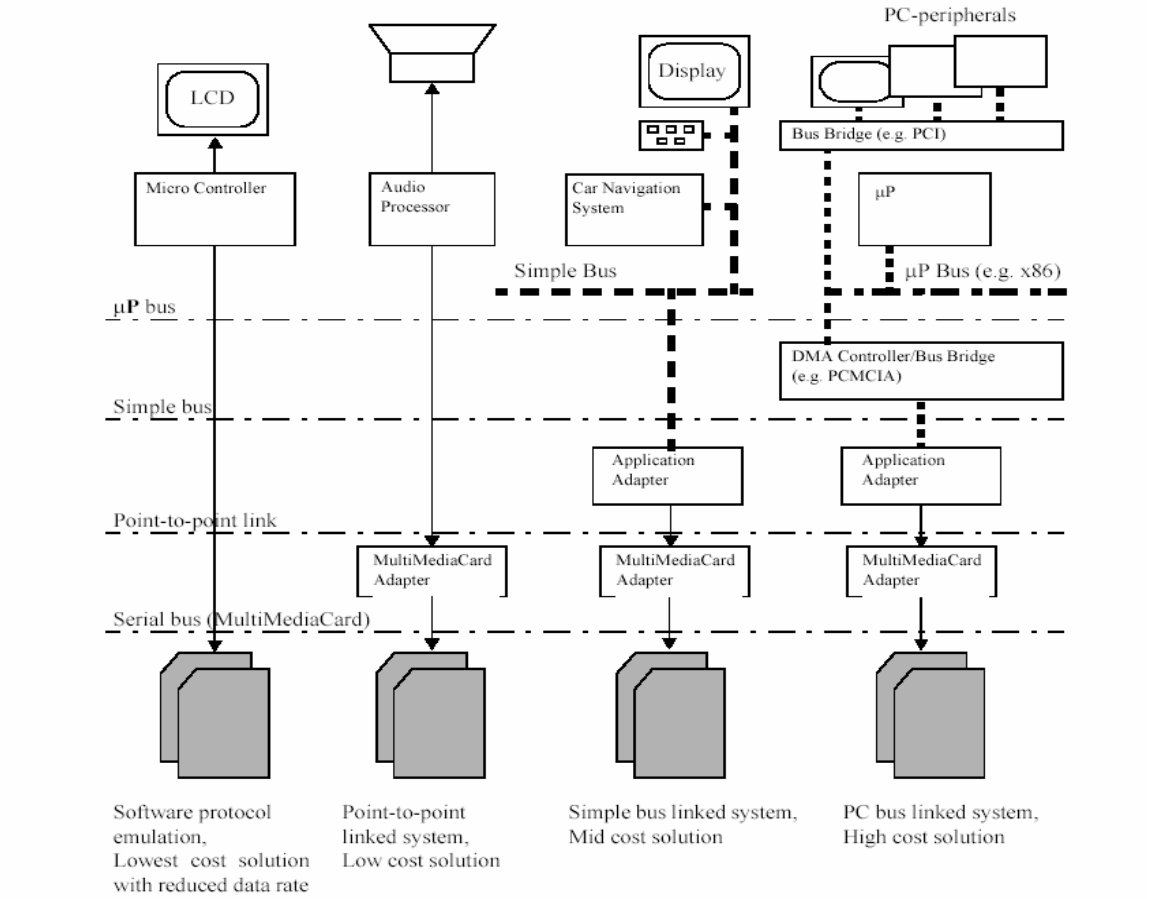
3.1.1 - Γενική Περιγραφή

Η MMC καθώς και η RS-MMC, η οποία χρησιμοποιεί ακριβώς το ίδιο interface, είναι πολύ μικρές , κινητές μονάδες αποθήκευσης οι οποίες σχεδιάστηκαν για εφαρμογές αποθήκευσης με πρωταρχικό σκοπό το μικρό μέγεθος, την μικρή κατανάλωση ισχύος και το χαμηλό κόστος αγοράς. Για τους λόγους αυτούς είναι ιδανικό μέσο αποθήκευσης για φορητές συσκευές που χρησιμοποιούν μπαταρίες. Βασικό χαρακτηριστικό της είναι η χαμηλή κατανάλωση ισχύος ενώ δεν χρειάζεται τροφοδοσία για να διατηρήσει τα αποθηκευμένα αρχεία. Ακόμα έχει μεγάλο εύρος σωστής λειτουργίας όσον αφορά την θερμοκρασία και την αντοχή σε κίνηση.

Όλα αυτά τα χαρακτηριστικά συνδυάζονται πολύ καλά σε ένα πολύ μικρό πακετάρισμα, για να ικανοποιήσουν τις ανάγκες μίας φορητής, χαμηλής κατανάλωσης, ηλεκτρονικής συσκευής. Το μέγεθος μίας MMC είναι 32mm x 24mm ενώ το πάχος της δεν ξεπερνά τα 1,4mm, ενώ επιπρόσθετα η RS-MMC είναι ακόμα μικρότερη με διαστάσεις 18mm x 24mm και πάχος 1,4mm, κάτι που την κάνει ιδανική για ένα μεγάλο πλήθος από φορητές συσκευές όπως MP3 players, τηλεφωνητές, ψηφιακές φωτογραφικές μηχανές, ηλεκτρονικά παιχνίδια, PDAs, κάμερες και πολλά άλλα.



Εικόνα 3.3 : Διαστάσεις MMC

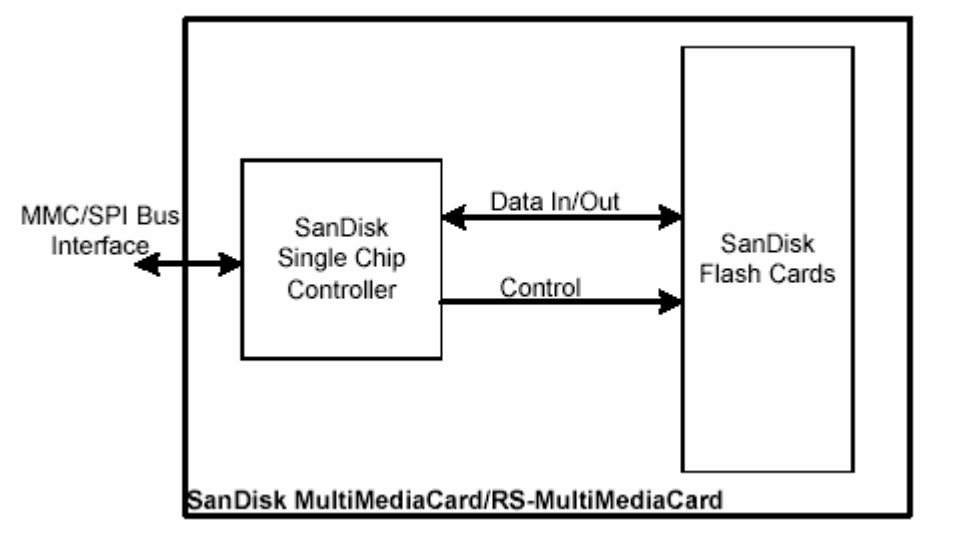


Εικόνα 3.4 : Τοπολογία συστημάτων που χρησιμοποιούν MMCs

Παρόλο τις πολλές εφαρμογές που μπορεί να χρησιμοποιηθεί, συνέπεια των χαρακτηριστικών της η κάρτα χρησιμοποιεί ένα απλό εφτά εισόδων – εξόδων σειριακό interface, το οποίο παρέχει πολλές δυνατότητες διαχείρισης στον προγραμματιστή. Όλα τα στοιχεία που αφορούν την κάρτα (όπως μέγιστη συχνότητα λειτουργίας, ταυτότητα της κάρτας κ.α.) είναι αποθηκευμένα στην ίδια την κάρτα.

Η κάρτα προσφέρει μεγάλη ελευθερία στην σχεδίαση συστημάτων ανεξάρτητα με τον μικροϋπολογιστή που χρησιμοποιείται. Για την συμβατότητα με τους υπάρχοντες μικροϋπολογιστές που υπάρχουν στο εμπόριο σήμερα, η κάρτα προσφέρει, εκτός και ένα εναλλακτικό interface επικοινωνίας, το οποίο είναι βασισμένο στο SPI.

Οι χωρητικότητες των MMC της SanDisk σήμερα φτάνουν στα 2 GB, βασισμένα σε ολοκληρωμένα ειδικά σχεδιασμένα για αυτή την εφαρμογή. Σε αντίθεση με αυτά τα ολοκληρωμένα η MMC περιέχει ένα «έξυπνο» μικροελεγκτή ο οποίος ελέγχει τα πρωτόκολλα επικοινωνίας, την αποθήκευση και την ανάγνωση δεδομένων καθώς και τους ECC (Error Correction Code) αλγόριθμους, την ρύθμιση της τροφοδοσίας και τον έλεγχο του ρολογιού.



Εικόνα 3.5 : Τοπολογία κατασκευής MMC

3.1.2 – Χαρακτηριστικά - Αρχιτεκτονική

Παρακάτω παρατίθενται τα χαρακτηριστικά της κάρτας όπως παρέχονται από την SanDisk Co.

- Υποστηρίζεται MultiMedia Card πρωτόκολλο
- Υποστηρίζεται SPI επικοινωνία
- Τάση τροφοδοσίας και λειτουργίας 2,7 μέχρι 3,6 Volt
- Μέγιστος ρυθμός μεταφοράς για μέχρι 10 κάρτες στον ίδιο διάδρομο
- Διόρθωση για λάθη σε περιοχές της μνήμης
- Εσωτερικές λειτουργίες προστασίας εγγραφής
- Μεταβλητό ρολόι 0-20 MHz
- Δυνατότητα τοποθέτησης πολλών καρτών στον ίδιο διάδρομο

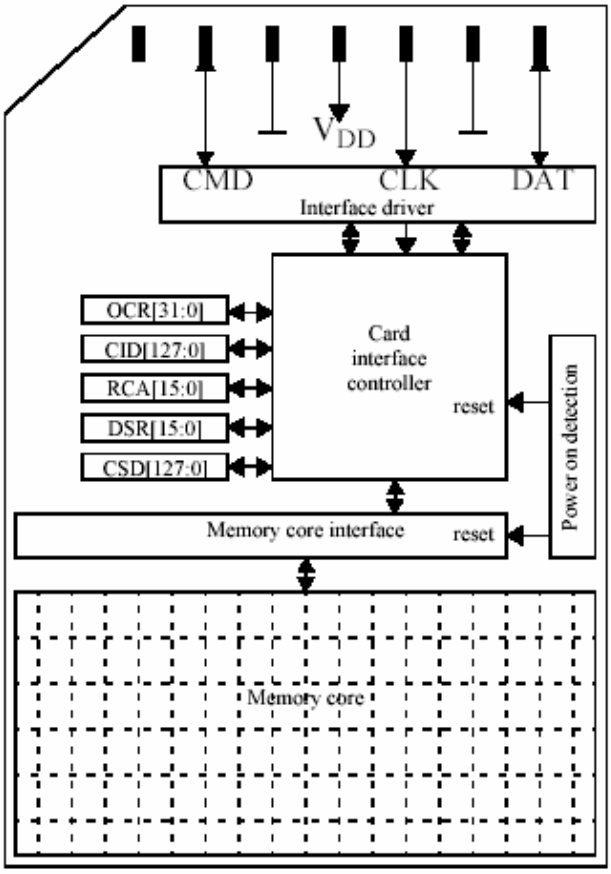
Τα παραπάνω χαρακτηριστικά της κάρτας δεικνύουν τις μεγάλες δυνατότητες που προσφέρει η MMC για τον σχεδιασμό του συστήματος.

Ακόμα παρατίθενται ορισμένα πλεονεκτήματα που προσφέρει ο εσωτερικός «έξυπνος» μικροελεγκτής που περιέχει η MMC.

- Ανεξαρτησία του μέσου το οποίο ελέγχει την κάρτα με τις λεπτομέρειες όσον αφορά την εγγραφή και τον προγραμματισμό της μνήμης
- Αναπτυγμένο σύστημα για την αναγνώριση και την διόρθωση λαθών
- Έλεγχος της τάσης για χαμηλή κατανάλωση

Ειδικά για το τελευταίο ας κάνουμε λίγο μνεία παραπάνω σε μία λειτουργία που έχει η MMC για την εξοικονόμηση ενέργειας. Η κάρτα εισέρχεται και εξέρχεται αυτόματα σε sleep κατάσταση αμέσως μόλις τελειώσει μία εντολή αν δεν λάβει καινούργια σε διάστημα 5 ms. Ο χρήστης δεν χρειάζεται να κάνει τίποτα για να μπει η κάρτα σε αυτή την κατάσταση ώστε να εξοικονομηθεί ενέργεια. Επιπλέον η κάρτα επιστρέφει από αυτή την κατάσταση αμέσως μόλις λάβει την επόμενη εντολή.

Η αρχιτεκτονική στην οποία είναι βασισμένη η σχεδίαση της MMC φαίνεται στο παρακάτω σχήμα.



Εικόνα 3.6 : Αρχιτεκτονική σχεδίασης MMC

Όπως φαίνεται και στην εικόνα οι απαιτήσεις στην σχεδίαση ήταν να κρατηθεί σε ένα ελάχιστο δυνατό οι απαραίτητες γραμμές για τον έλεγχο της κάρτας. Έτσι η κάρτα μπορεί να ελεγχθεί μόνο με τρία σήματα όταν είναι σε MultiMedia λειτουργία, και ένα επιπρόσθετο σήμα για την επιλογή της σε SPI λειτουργία. Τα σήματα φαίνονται παρακάτω:

- **CLK:** με κάθε κύκλο του σήματος αυτού (ρολογιού) ένα bit στις γραμμές εντολών και δεδομένων μεταφέρεται. Η συχνότητα του ρολογιού μπορεί να διαφέρει από 0 έως το μέγιστο που μπορεί να υποστηρίξει η κάρτα.
- **CMD:** είναι μία διπλής κατευθύνσεως γραμμή για μεταφορά των δεδομένων των εντολών από την κάρτα στον master του διαδρόμου και αντίθετα.

- **DAT:** είναι μία διπλής κατευθύνσεως γραμμή για την μεταφορά δεδομένων από την κάρτα στον master. Μόνο μία κάρτα μπορεί να χρησιμοποιεί την γραμμή κάθε φορά.

Μεγαλύτερη αναφορά θα γίνει κατά την περιγραφή των δύο ειδών λειτουργίας της κάρτας

Η MMC όπως έχει ήδη αναφερθεί έχει εφτά επαφές στην μία πλευρά της. Η περιγραφή τους φαίνεται στο παρακάτω πίνακα:

Πίνακας 3.1 : Περιγραφή εισόδων - εξόδων MMC

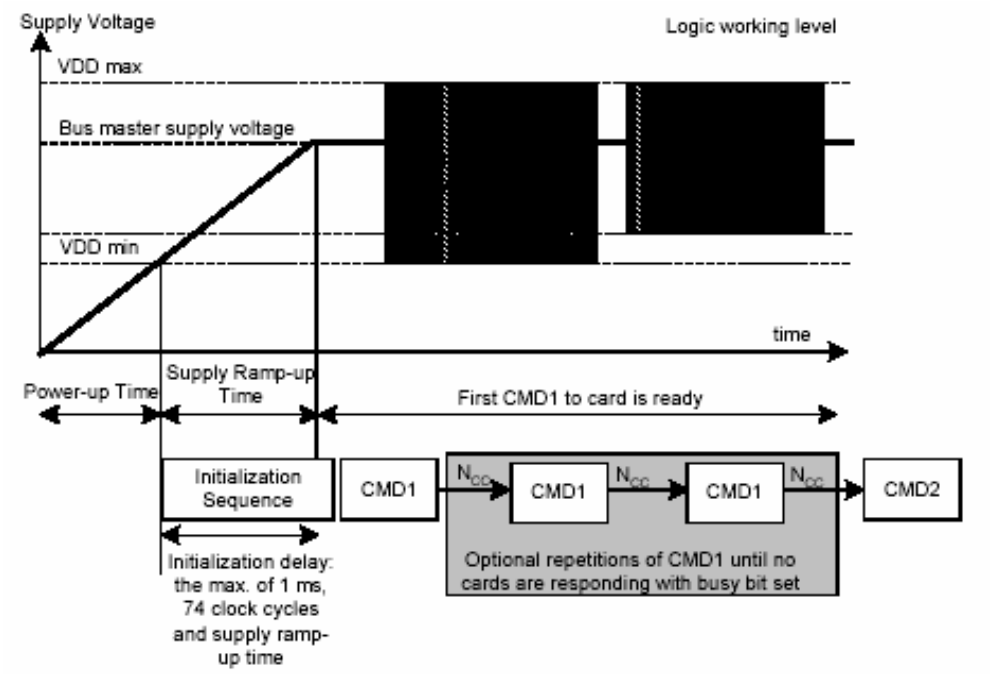
Pin No.	Name	Type[1]	Description
MultiMediaCard Mode			
1	RSV	NC	Not connected or Always "1"
2	CMD	I/O, PP, OD	Command/Response
3	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground
7	DAT0	I/O, PP	Data 0
SPI Mode			
1	CS	I	Chip Select (active low)
2	DataIn	I	Host-to-card Commands and Data
3	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground
7	DataOut	O	Card-to-host Data and Status

Ας αναφερθούμε τώρα στους καταχωρητές που διαθέτει η κάρτα. Κατ' αρχάς υπάρχει ο OCR (Operating Conditions Register). Αυτός είναι ένας 32-bit καταχωρητής ο οποίος περιέχει την τάση τροφοδοσίας της κάρτας. Επιπρόσθετα περιέχει ένα bit το οποίο τίθεται όταν τελειώσει η τροφοδότηση της κάρτας. Στην συνέχεια, υπάρχει ο CID καταχωρητής (Card Identification Register) ο οποίος έχει μέγεθος 128 bits. Αυτός περιέχει την ταυτοποίηση της κάρτας, προγραμματίζεται κατά την κατασκευή της κάρτας και τα περιεχόμενα δεν μπορούν να αλλάξουν έπειτα. Ακόμα υπάρχει ο 128-bit καταχωρητής CSD (Card Specific Data) ο οποίος περιέχει όλες τις απαραίτητες ρυθμίσεις για να έχει την δυνατότητα ο χρήστης να διαβάζει τα δεδομένα. Τέλος να αναφερθούμε στον RCA

(Relative Card Address) μεγέθους 16-bit που κρατάει την διεύθυνση που ανατίθεται στην κάρτα από το σύστημα, και στον ίδιου μεγέθους DSR (Driver Stage Register) ο οποίος μπορεί να χρησιμοποιηθεί για την βελτίωση κάποιων χαρακτηριστικών λειτουργίας της κάρτας. Τέλος υπάρχει ο 32-bit CSR (Card Status Register) ο οποίος περιέχει την κατάσταση που βρίσκεται η κάρτα. Από αυτούς τους καταχωρητές σε SPI λειτουργία μόνο οι CSD και CID είναι προσβάσιμοι.

3.1.3 – Αρχικοποίηση

Μετά την τροφοδότηση της η κάρτα (ή μία επανεκκίνηση της με χρησιμοποίηση της εντολής 0) πρέπει να ακολουθηθεί μία συγκεκριμένη διαδικασία για την αρχικοποίηση της κάρτας. Αυτή η ακολουθία φαίνεται παρακάτω:



Εικόνα 3.7 : Διαδικασία αρχικοποίησης MMC

Μετά την τροφοδότηση οι κάρτες που βρίσκονται πάνω στον διάδρομο αγνοούν κάθε μετάδοση μέχρι να λάβουν την εντολή 1. Αυτή η εντολή είναι μία ειδική εντολή συγχρονισμού η οποία χρησιμοποιείται για την διαπραγμάτευση της τάσης λειτουργίας της κάρτας και τον έλεγχο της κάρτας έως ότου αυτή έχει τελειώσει την διαδικασία αρχικοποίησης. Η εντολή 1 περιέχει μία σημαία η οποία όταν έχει τεθεί δηλώνει ότι η κάρτα δεν έχει τελειώσει την διαδικασία αρχικοποίησης της. Τότε ο master πρέπει να περιμένει συνεχίζοντας να στέλνει την εντολή 1 μέχρι όλες οι κάρτες να είναι έτοιμες για επικοινωνία. Αυτή η διαδικασία έχει μέγιστο χρόνο ολοκλήρωσης 500ms.

Ο master του διαδρόμου έχει την υποχρέωση να οδηγήσει και κάθε κάρτα ξεχωριστά αλλά και το όλο σύστημα καρτών να τελειώσει την διαδικασία αρχικοποίησης. Επειδή ο χρόνος τροφοδότησης και ο χρόνος ανύψωσης της τροφοδοσίας εξαρτάται από πολλές παραμέτρους του συστήματος, όπως το μήκος του διαδρόμου, το πλήθος των καρτών και το είδος της τροφοδοσίας, ο master πρέπει να βεβαιωθεί ότι ο χρόνος που χρειάζεται η τάση για να φτάσει στα επιθυμητά επίπεδα έχει περάσει προτού μεταδώσει την εντολή 1.

Έτσι μετά την τροφοδότηση ο master του διαδρόμου αρχίζει την διαδικασία αρχικοποίησης . Αυτή είναι ένα ρεύμα από λογικά “1” στον διάδρομο. Το μέγιστο αυτής

της διαδικασίας είναι το μέγιστο από 1ms και 74 κύκλους ρολογιού (10 παραπάνω κύκλους από αυτό που υποτίθεται χρειάζεται η κάρτα για την ολοκλήρωση της αρχικοποίησης της.

3.1.4 – Περιγραφή διαθέσιμων interface επικοινωνίας με την MMC

Όπως έχει ήδη αναφερθεί η MMC υποστηρίζει δύο πρωτόκολλα επικοινωνίας. Παρακάτω γίνεται αναφορά στα δύο interface, δίνοντας ιδιαίτερη έμφαση στο SPI interface, το οποίο και χρησιμοποιήθηκε, λόγω της ευκολίας στη χρήση του αλλά και της πλήρους συμβατότητας με τον μικροελεγκτή ATmega16 της ATMEL.

3.1.4.1 – MultiMedia Card λειτουργία

Σε αυτό το είδος λειτουργίας ο master του διαδρόμου ελέγχει την επικοινωνία με τις MMC. Υπάρχουν δύο είδη εντολών σε αυτό το είδος λειτουργίας:

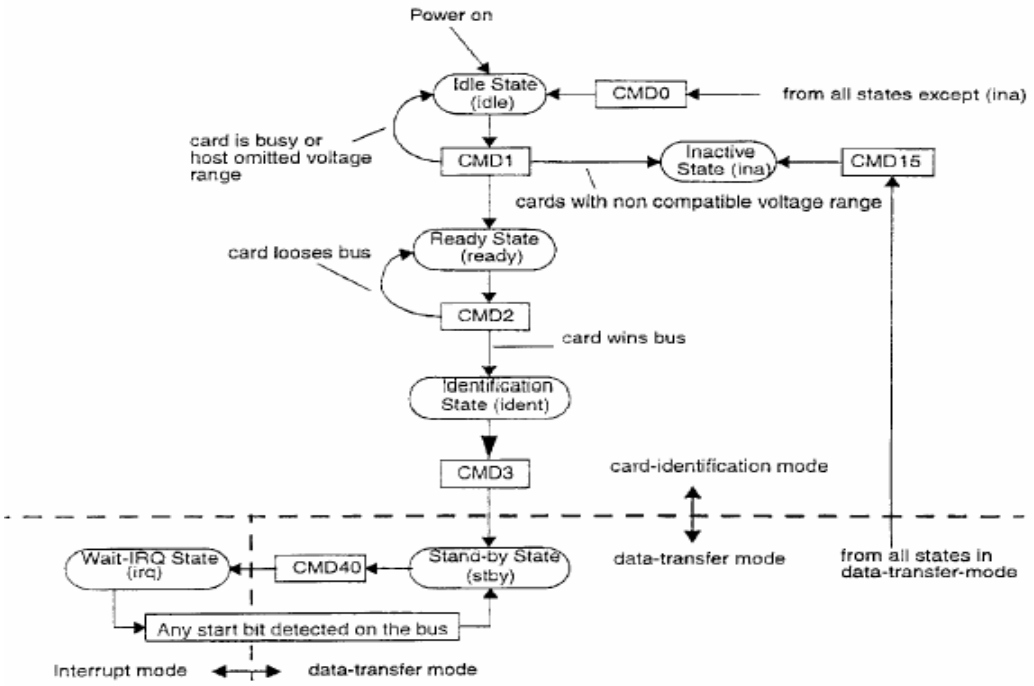
1. **Broadcast εντολές** – Αυτές οι εντολές απευθύνονται σε όλες τις κάρτες που βρίσκονται πάνω στον διάδρομο. Μερικές απ' αυτές απαιτούν απάντηση.
2. **Addressed (point-to-point) εντολές** – Αυτές οι εντολές στέλνονται σε μία συγκεκριμένη διευθυνσιοδοτημένη κάρτα και απαιτούν την απάντηση από την συγκεκριμένη κάρτα.

Η MMC έχει δύο περιόδους λειτουργίας.

- Περίοδος αναγνώρισης κάρτας. Η κάρτα βρίσκεται σε αυτή την λειτουργία αφού έχει τροφοδοτηθεί και δεν έχει λάβει ακόμα την εντολή 3. Σε αυτή την περίοδο ο master του διαδρόμου ψάχνει ακόμα για νέες κάρτες στον διάδρομο.
- Περίοδος μεταφοράς δεδομένων. Η MMC εισέρχεται σε αυτή την λειτουργία όταν λάβει μία RCA. Ο master θα μπει σε αυτή την λειτουργία όταν αναγνωρίσει όλες τις κάρτες στον διάδρομο.

3.1.4.1.1 – Περίοδος αναγνώρισης κάρτας

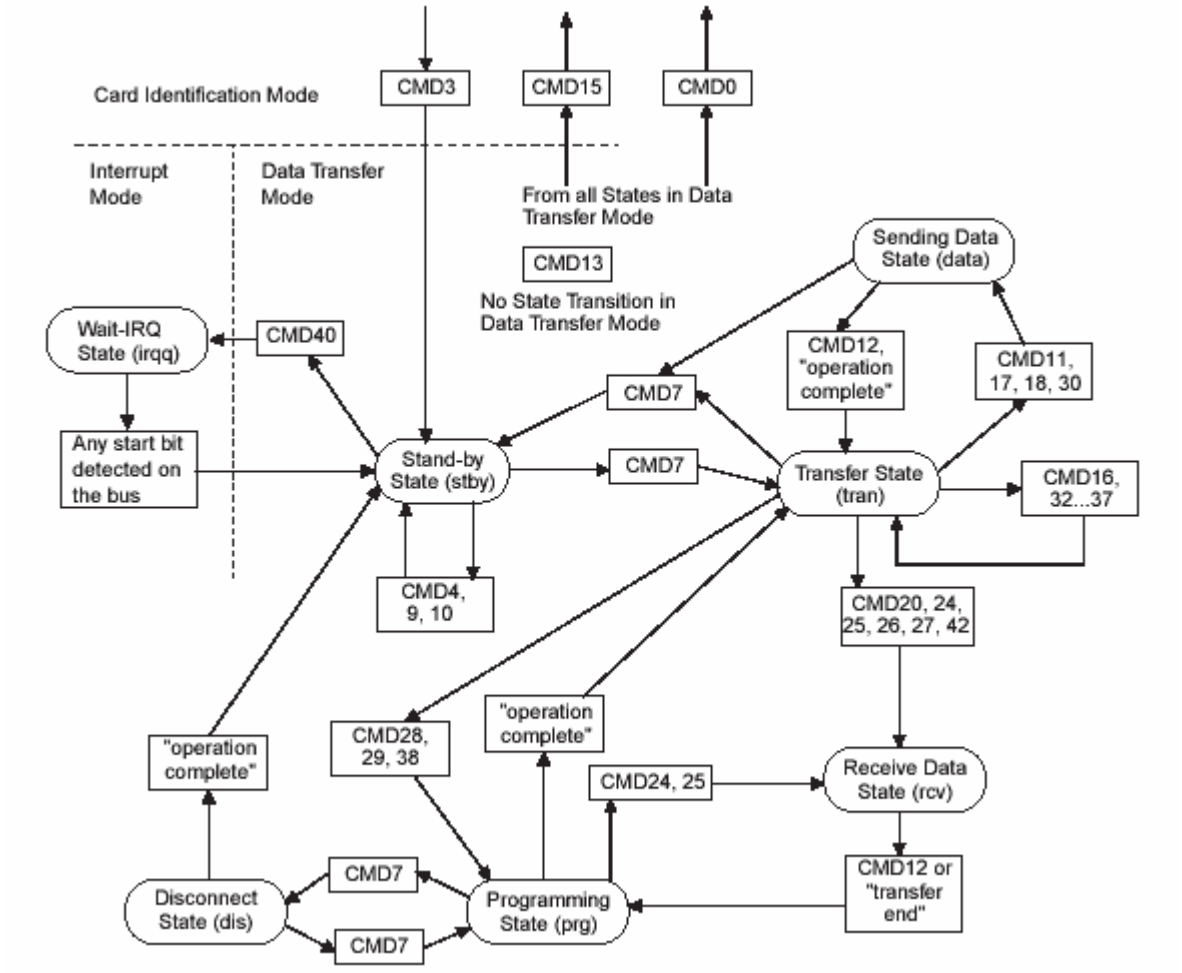
Κατά την περίοδο αυτή ο master επανεκκινεί όλες τις κάρτες οι οποίες βρίσκονται σε αυτή την λειτουργία, επικυρώνει το επίπεδο τάσης λειτουργίας, αναγνωρίζει τις κάρτες και τους ζητάει να μεταδώσουν μία RCA. Σε αυτή την περίοδο όλες οι μεταφορές δεδομένων γίνονται στην CMD γραμμή μόνο.



Εικόνα 3.8 : Διάγραμμα καταστάσεων MMC σε περίοδο αναγνώρισης καρτών

3.1.4.1.2 – Περίοδος μεταφοράς δεδομένων

Κατά την περίοδο λειτουργίας αυτής οι CMD και DAT γραμμές είναι σε push-pull κατάσταση. Παρακάτω φαίνεται το διάγραμμα καταστάσεων της κάρτας σε αυτή την περίοδο λειτουργίας.



Εικόνα 3.9 : Διάγραμμα καταστάσεων MMC σε περίοδο μεταφοράς δεδομένων

Ας αναφερθούμε τώρα λίγο στην διαδικασία ανάγνωσης και εγγραφής δεδομένων. Κατά την ανάγνωση, η μετάδοση αρχίζει με ένα bit σε χαμηλό λογικό επίπεδο – ενώ η γραμμή όταν δεν υπάρχει μετάδοση πάντα διατηρείται σε υψηλό επίπεδο - και τελειώνει με ένα υψηλού λογικού επιπέδου bit. Η μετάδοση περιέχει τα δεδομένα και bits διόρθωσης λαθών. Η διαδικασία κατά την εγγραφή είναι παρόμοια.

Είναι δυνατό ο master να μειώσει ή ακόμα και να κλείσει το ρολόι στον διάδρομο για να εξοικονομηθεί ενέργεια. Τέλος να αναφέρουμε ότι η κάρτα μπορεί να κλειδωθεί και να ξεκλειδωθεί , ενώ παρέχει και ένα αλγόριθμο αναγνώρισης και διόρθωσης λαθών κατά την μετάδοση στον διάδρομο με χρησιμοποίηση του ελέγχου CRC (Cyclic Redundancy Codes).

3.1.4.2 –SPI λειτουργία

Το SPI πρωτόκολλο επικοινωνίας που χρησιμοποιήθηκε στην εργασία είναι ένα δεύτερο πρωτόκολλο επικοινωνίας που προσφέρει η κάρτα και χρησιμοποιείται για την εύκολη επικοινωνία της κάρτας μέσω ενός SPI καναλιού που πρόσφατα προστέθηκε ε ορισμένους μικροελεγκτές όπως και σε ορισμένα μοντέλα του AVR Αυτό το πρωτόκολλο είναι ένα υποσύνολο του προηγούμενου πρωτοκόλλου. Η επιλογή του τρόπου επικοινωνίας με την κάρτα επιλέγεται κατά την επανεκκίνηση της κάρτας (εντολή 0), η κάρτα μπαίνει σε SPI λειτουργία αν είσοδος επιλογής της κρατηθεί σε χαμηλό λογικό επίπεδο κατά την λήψη της εντολής. Αυτό το πρωτόκολλο επικοινωνίας πλεονεκτεί όσον αφορά το προηγούμενο όσον αφορά ότι μπορεί να χρησιμοποιήσει μεγάλο εύρος συσκευών για master ενώ παράλληλα κρατάει την προσπάθεια στον σχεδιασμό στο ελάχιστο δυνατό. Από την άλλη όμως παρέχει μικρότερες ταχύτητες επικοινωνίας, απαιτεί ξεχωριστό σήμα επιλογής για κάθε κάρτα στον διάδρομο και περιορίζει τον μέγιστο αριθμό καρτών στον ίδιο διάδρομο.

3.1.4.2.1 – Σκεπτικό δημιουργίας SPI λειτουργίας

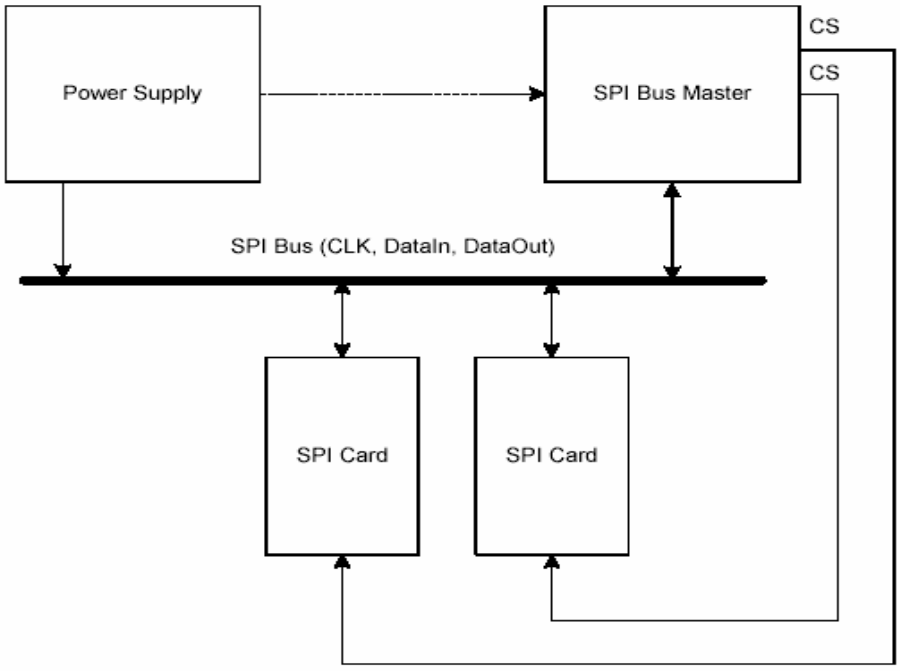
Το SPI όπως έχει ήδη προαναφερθεί ανακαλύφθηκε από την MOTOROLA και πλέον υπάρχει σε πολλούς vendor μικροελεγκτές. Το πρωτόκολλο παρέχει αξιόπιστη σειριακή επικοινωνία σε μεγάλες ταχύτητες βασισμένο πάνω σε τέσσερα σήματα μόνο:

- CS (Chip Select) – Σήμα επιλογής από τον master στην κάρτα
- CLK (Clock) – Σήμα ρολογιού από τον master στην κάρτα
- DI (Data In) – Δεδομένα από τον master στην κάρτα
- DO (Data Out) – Δεδομένα από την κάρτα στον master

Άλλο ένα χαρακτηριστικό του πρωτοκόλλου είναι ότι όλες οι μεταδόσεις είναι πολλαπλάσια του byte (8 bits) και πάντα συγχρονισμένα με το σήμα του ρολογιού.

3.1.4.2.2 – Τοπολογία SPI καναλιού

Σε αυτό το είδος λειτουργίας η αναγνώριση της κάρτας και η διευθυσιοδότηση των καρτών έχουν αντικατασταθεί με το σήμα hardware CS (Chip Select), ενώ επιπλέον δεν υπάρχουν εντολές που να απευθύνονται σε όλες τις κάρτες στο κανάλι. Για κάθε εντολή η κάρτα – slave επιλέγεται θέτοντας σε λογικό 0 το CS σήμα της (επιλογή της κάρτας σημαίνει CS λογικά χαμηλό)



Εικόνα 3.10 : Τοπολογία καναλιού SPI

Το σήμα επιλογής της κάρτας πρέπει να είναι συνεχώς ενεργό (λογικά χαμηλό επίπεδο) κατά την διάρκεια της SPI επικοινωνίας με την κάρτα (εντολή, απάντηση και αποστολή δεδομένων). Το μόνο χρονικό σημείο που μπορεί το σήμα επιλογής να μην είναι ενεργό, είναι κατά την διάρκεια του προγραμματισμού της κάρτας.

Στην SPI λειτουργία οι διπλής κατεύθυνσης γραμμές CMD και DAT έχουν αντικατασταθεί από τις μονής κατευθύνσεως DI (DataIn) και DO (DataOut). Αυτό αποκλείει την δυνατότητα εκτέλεσης εντολών κατά την διάρκεια της μεταφοράς δεδομένων. Συνεπώς δεν υπάρχει δυνατότητα συνεχόμενης σειριακής μεταφοράς δεδομένων και μεταφοράς πολλαπλών τμημάτων δεδομένων. Άρα η μόνη λειτουργία που υποστηρίζεται είναι η μεταφορά ενός μόνο η πολλαπλών block την φορά. Τέλος το SPI χρησιμοποιεί τα σήματα που περιγράφηκαν σε παραπάνω πίνακα.

3.1.4.2.3 – Καταχωρητές σε SPI λειτουργία

Στην SPI λειτουργία δεν είναι διαθέσιμοι όλοι οι καταχωρητές που είναι διαθέσιμοι στην MultiMedia λειτουργία. Παρακάτω φαίνεται ένας πίνακας με τους προσβάσιμους καταχωρητές σε SPI λειτουργία.

Πίνακας 3.2 : Περιγραφή καταχωρητών σε SPI

Name	Available in SPI mode	Width [Bytes]	Description
CID	Yes	16	Card identification data (serial number, manufacturer ID, etc.).
RCA	No		
DSR	No		
CSD	Yes	16	Card-specific data, information about the card operation conditions.
OCR	Yes	32	Operation condition register.

3.1.4.2.4 – Πρωτόκολλο SPI καναλιού για MMC

Ενώ το προηγούμενο πρωτόκολλο είναι βασισμένο σε ρεύματα bits τα οποία αρχίζουν με ένα bit εκκίνησης και τελειώνουν με ένα bit τερματισμού τόσο για τις εντολές τόσο και για τα δεδομένα, αυτό το πρωτόκολλο είναι αυτό που λέμε byte oriented. Δηλαδή κάθε μεταφορά είναι πολλαπλάσιο του byte ενώ επιπλέον η μετάδοση είναι ταυτόχρονη με την ενεργοποίηση του CS σήματος.

Παρόμοια με τα προηγούμενα, όλα τα σήματα αποτελούνται από εντολές, απαντήσεις και κομμάτια δεδομένων. Ο master ελέγχει όλη την επικοινωνία στον διάδρομο. Αυτός αρχίζει και την επικοινωνία ενεργοποιώντας το CS σήμα.

Το είδος της απάντησης διαφέρει στα εξής σημεία με τα προηγούμενα.

- Οι κάρτες πάντα απαντούν στις απαιτήσεις του master
- Χρησιμοποιούνται δομές απαντήσεων μεγέθους (8, 16 και 40 bits)
- Η κάρτα αν συναντήσει κάποιο πρόβλημα κατά την επεξεργασία των εντολών απαντά ότι συνέβη κάποιο λάθος και δεν χρησιμοποιείται κάποια λήξη χρονικής προθεσμίας όπως στο MultiMedia πρωτόκολλο.

3.1.4.2.5– Επιλογή λειτουργίας της MMC με SPI

Οι MMC “ξυπνούν” σε MultiMedia λειτουργία. Η MMC θα μπει σε SPI λειτουργία αν το σήμα είναι ενεργό (λογικό 0) κατά την διάρκεια της εντολής 0. Η επιλογή του SPI δεν είναι απαραίτητο να γίνει μόνο κατά αυτή την χρονική περίοδο. Κάθε φορά που λαμβάνεται η εντολή 0 δειγματοληπτείται το σήμα CS και αν είναι λογικό 0 τότε η κάρτα μπαίνει σε SPI λειτουργία. Ο μόνος τρόπος να γυρίσει η κάρτα σε MultiMedia λειτουργία είναι μέσω

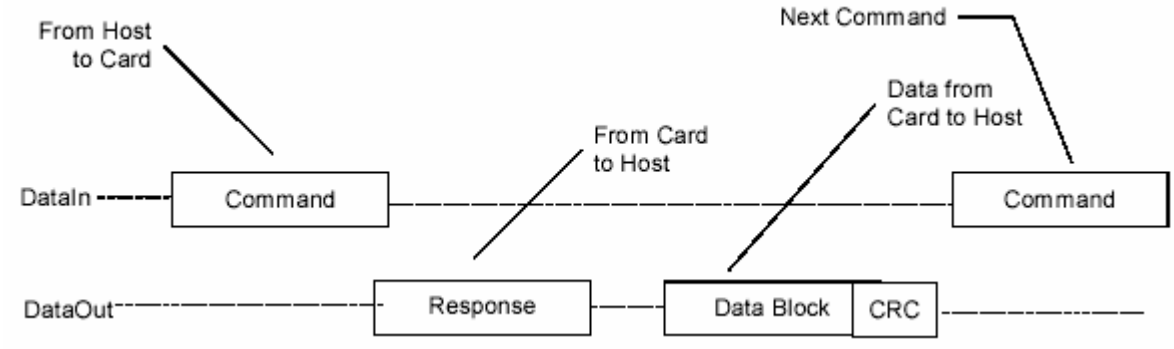
ενός κύκλου της τροφοδοσίας (κλείσιμο και άνοιγμα της τροφοδοσίας). Τέλος να αναφερθεί ότι αν η κάρτα διαπιστώσει ότι χρειάζεται να μείνει σε MultiMedia λειτουργία δεν θα απαντήσει στην εντολή 0 και θα παραμείνει στην ίδια κατάσταση.

3.1.4.2.6– Ασφάλεια μεταφοράς δεδομένων στο SPI κανάλι

Ασφάλεια μεταφοράς δεδομένων δεν προσφέρεται στην SPI λειτουργία, αφήνεται πάνω στον σχεδιαστή να δημιουργήσει ασφαλή μέσα για την επικοινωνία και αλγόριθμους διόρθωσης δεδομένων. Η μόνη εντολή που χρησιμοποιεί CRC είναι η εντολή 0 (τότε ακόμα η κάρτα είναι σε MultiMedia λειτουργία) και για αυτό το CRC για αυτή πρέπει να είναι ίσο με 0x95.

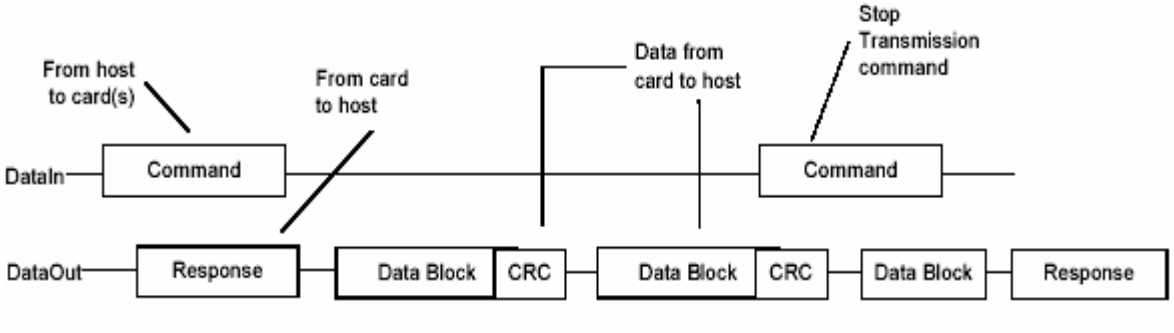
3.1.4.2.7– Ανάγνωση δεδομένων

Η SPI λειτουργία υποστηρίζει ανάγνωση ενός ή πολλαπλών block δεδομένων. Η βασική διαφορά με το MultiMedia πρωτόκολλο είναι ότι τόσο η απάντηση της κάρτας όσο και τα δεδομένα μεταφέρονται στην γραμμή DO. Έτσι μπορεί η απάντηση της κάρτας στην εντολή 12 (Stop_Transmission) να αντικαταστήσει το τελευταίο κομμάτι δεδομένων. Παρακάτω φαίνεται η μορφή και ο χρονισμός της μετάδοσης δεδομένων κατά την εγγραφή δεδομένων.



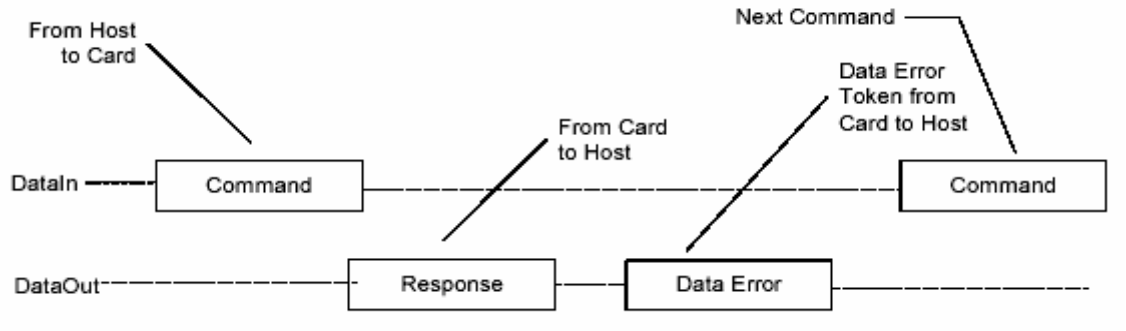
Εικόνα 3.11 : Ανάγνωση ενός block δεδομένων

Το block είναι η βασική μονάδα για ανάγνωση και εγγραφή δεδομένων, του οποίου το μέγεθος αλλάζει με ειδική εντολή (προκαθορισμένη τιμή είναι 512 bytes). Υπάρχει δυνατότητα να διαβαστεί και μέρος block δεδομένων αν έχει προηγηθεί ειδική εντολή. Επιπλέον υπάρχει δυνατότητα ανάγνωσης πολλαπλών block.



Εικόνα 3.12 : Ανάγνωση πολλαπλών block δεδομένων

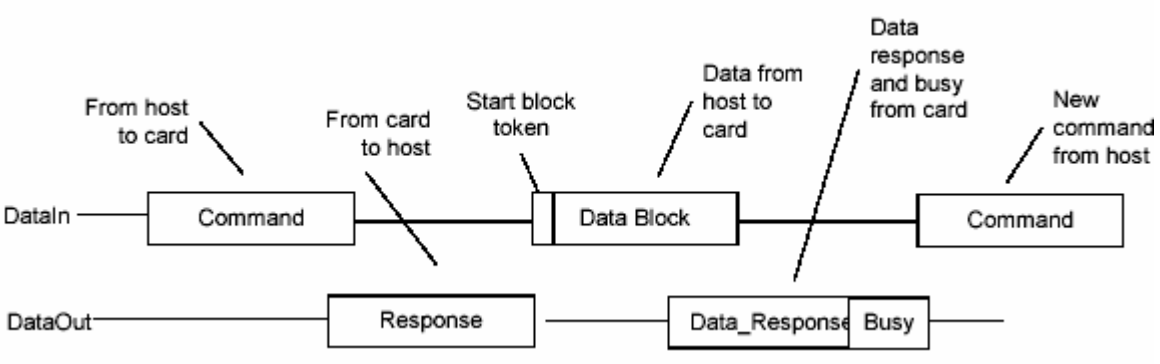
Αν συμβεί κάποιο λάθος κατά την ανάγνωση η κάρτα δεν θα στείλει δεδομένα αλλά μία ειδική απάντηση που θα δεικνύει το λάθος που συνέβη.



Εικόνα 3.13 : Λάθος στην ανάγνωση δεδομένων

3.1.4.2.8– Εγγραφή δεδομένων

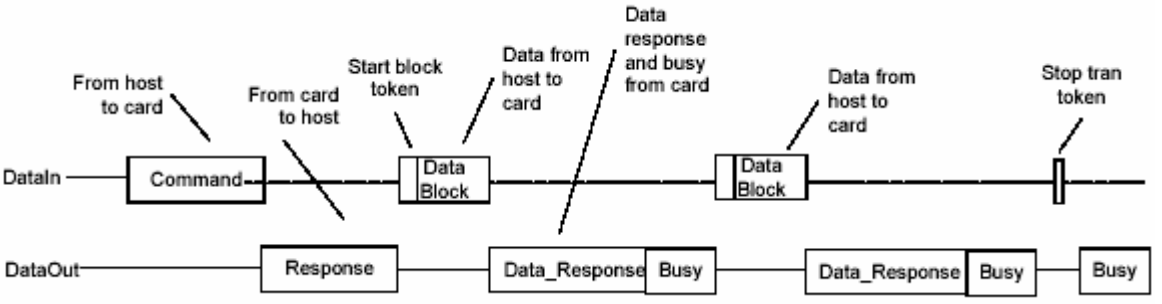
Παρόμοια με την ανάγνωση σε αυτό το πρωτόκολλο, υποστηρίζεται εγγραφή ενός ή πολλαπλών block δεδομένων. Με την λήψη μίας έγκυρης εντολής εγγραφής (ενός ή πολλαπλών block) η κάρτα θα στείλει την κατάλληλη απάντηση και θα περιμένει τα block που θα προγραμματιστούν. Αν συμβεί κάποιο λάθος θα ενημερωθεί η απάντηση και τα δεδομένα που τυχόν θα παραληφθούν θα αγνοηθούν.



Εικόνα 3.14 : Εγγραφή ενός block δεδομένων

Καθενός block προηγείται ένα byte (Start Block Token). Μετά από την λήψη κάθε block η κάρτα απαντά και αν δεν υπάρχει κάποιο λάθος προγραμματίζει τα δεδομένα παραμένοντας σε busy κατάσταση.

Κατά την εγγραφή πολλαπλών block η κάρτα θα λαμβάνει block από δεδομένα μέχρι να μεταδοθεί ένα ειδικό byte (Stop Block Token). Το πλήθος των block που μεταδίδονται σε μία πολλαπλή μετάδοση δεν ορίζεται πουθενά απλά σταματά με την μετάδοση του byte τερματισμού μετάδοσης.



Εικόνα 3.15 : Εγγραφή πολλαπλών block δεδομένων

Όταν αναγνωριστεί κάποιο λάθος στην εγγραφή η κάρτα θα γνωστοποιήσει την αποτυχία και θα αγνοήσει τα επόμενα block. Τότε ο master πρέπει να στείλει την εντολή

τερματισμού μετάδοσης δεδομένων. Ακόμα όταν η εγγραφή τελειώσει πρέπει να ελέγξει την κατάσταση της κάρτας.

3.1.4.2.9– Διαδικασία αρχικοποίησης

Η αρχικοποίηση της κάρτας είναι παρόμοια με την προηγούμενη λειτουργία. Η κάρτα μετά την τροφοδότηση της μπαίνει σε idle κατάσταση και οι μόνες έγκυρες εντολές είναι οι εντολές 1 και 58. Η εντολή πρέπει να σταλθεί επανειλημμένα στην κάρτα μέχρι η σημαία busy να γίνει ίση με μηδέν, δηλώνοντας έτσι ότι η κάρτα τελείωσε την αρχικοποίηση της και είναι έτοιμη να δεχτεί εντολές. Η εντολή 58 χρησιμοποιείται για να επιστρέψει το περιεχόμενο του OCR (σε SPI λειτουργία αυτός δεν επιστρέφεται με την εντολή 1). Περισσότερα για την αρχικοποίηση αναφέρονται παρακάτω.

3.1.4.2.10–Έλεγχος ρολογιού στο κανάλι

Όπως και στο προηγούμενο πρωτόκολλο το ρολόι του διαδρόμου (που είναι υπευθυνότητα του master να το παρέχει) μπορεί να ρυθμιστεί για την εξοικονόμηση ενέργειας και να αποφευχθούν καταστάσεις υπερχείλισης ή έλλειψης δεδομένων. Ο master έχει το δικαίωμα να αλλάξει την συχνότητα του ρολογιού ή ακόμα και να το σταματήσει όμως κάτω από κάποιους περιορισμούς.

- Η μέγιστη συχνότητα καθορίζεται από τις δυνατότητες της κάρτας που δίνονται από τον κατασκευαστή.
- Το ρολόι πρέπει να τρέχει για την ολοκλήρωση των εργασιών της κάρτας

Μετά την τελευταία μετάδοση στο διάδρομο ο master πρέπει να παρέχει 8 κύκλους ρολογιού για να ολοκληρώσει η κάρτα τις εργασίες της, πριν σταματήσει το ρολόι. Σε αυτή την περίοδο δεν έχει σημασία η κατάσταση του CS σήματος.

- Ο master μπορεί να κλείσει το ρολόι σε μία κάρτα που είναι busy. Αυτή θα συνεχίσει την προγραμματιστική εργασία της χωρίς να χρειάζεται παλμούς. ¨όμως ο χρήστης πρέπει να της παρέχει μία ακμή ρολογιού για να βγει από αυτή την κατάσταση

3.1.4.2.11– Λίστα εντολών στην SPI λειτουργία

Τα παρακάτω παρέχουν χρήσιμες πληροφορίες για τις εντολές σε SPI λειτουργία.

3.1.4.2.11.1–Κλάσεις εντολών στην SPI λειτουργία

Οι κλάσεις των εντολών που είναι διαθέσιμες σε SPI λειτουργία φαίνονται στον παρακάτω πίνακα.

Πίνακας 3.3 : Κλάσεις εντολών σε SPI

	Class	Supported Commands																							
CCC	Descrip.	0	1	9	10	12	13	16	17	18	23	24	25	27	28	29	30	35	36	38	42	55	56	58	59
0	Basic	+	+	+	+		+																	+	+
1	NS																								
2	Block read					+		+	+	+	+														
3	NS																								
4	Block write							+			+	+	+	+											
5	Erase																	+	+	+					
6	Write-protect														+	+	+								
7	Lock card																				+				
8	App-specific																					+	+		
9	NS																								
10-11	R																								

3.1.4.2.11.2–Περιγραφή εντολών στην SPI λειτουργία

Οι εντολές έχουν μήκος 6 bytes (1 byte κωδικός, 4 byte όρισμα και 1 byte CRC). Όλα τα μνημονικά ονόματα των εντολών αντιστοιχούν στους κωδικούς τους με την προσθήκη ότι το 6^{o} bit είναι πάντα λογικό 1. Παρακάτω φαίνεται ένας πίνακας με τις εντολές της κάρτα (yes σημαίνει ότι η εντολή υποστηρίζεται σε SPI λειτουργία).

Πίνακας 3.4 : Λίστα εντολών σε SPI

CMD Index	SPI Mode	Argument	Resp	Abbreviation	Description
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets MultiMediaCard or RS-MultiMediaCard.
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD2	No	---	---	---	---
CMD3	No	---	---	---	---
CMD4	No	---	---	---	---
CMD5	Reserved				
CMD6	Reserved				
CMD7	No	---	---	---	---
CMD Index	**SPI Mode**	**Argument**	**Resp**	**Abbreviation**	**Description**
CMD8	Reserved				
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its identification (CID).
CMD11	No	---	---	---	---
CMD12	Yes	None	R1	STOP_ TRANSMISSION	Forces card to stop transmission during a multiple block read operation.
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD14	Reserved				
CMD15	No	---	---	---	---
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write).
CMD17	Yes	[31:0] data address	R1	READ_SINGLE_ BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE_ BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_TRANSMISSION command or the requested number of data blocks transmitted.

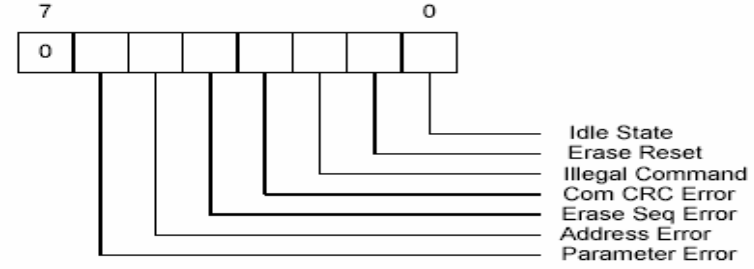
CMD19	Reserved				
CMD20	No	---	---	---	---
CMD21 … CMD23	Reserved				
CMD24	Yes	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_ BLOCK	Continuously writes blocks of data until a stop transmission token is sent or the requested number of blocks is received.
CMD26	No	---	---	---	---
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. Write protection properties are coded in the card-specific
CMD29	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group.
CMD33	Yes	[31:0] data address	R1	TAG_SECTOR_END	Sets the address of the last sector in a continuous range within the selected erase group, or the address of a single sector to be selected for erase.
CMD34	Yes	[31:0] data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection.
CMD35	Yes	[31:0] data address	R1	TAG_ERASE_GROUP _START	Sets the address of the first erase group within a range to be selected for erase.

CMD36	Yes	[31:0] data address	R1	TAG_ERASE_GROUP _END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	Yes	[31:0] data address	R1	UNTAG_ERASE_ GROUP	Removes one previously selected erase group from the erase selection.
CMD38	Yes	[31:0] stuff bits	R1b	ERASE	Erases all previously selected sectors.
CMD39	No	---	---	---	---
CMD40	No	---	---	---	---
CMD41	Reserved				
CMD42	Yes	[31:0] stuff bits	R1b	LOCK_UNLOCK	Set/resets the password or lock/unlock the card. The size of the Data Block is defined by the SET_BLOCK_LEN command.
CMD43 … CMD54	Reserved				
CMD55	Yes	This optional MMCA command is not supported in the SanDisk MultiMediaCard/RS-MultiMediaCard.			
CMD56	Yes	This optional MMCA command is not supported in the SanDisk MultiMediaCard/RS-MultiMediaCard.			
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR Register of a card.
CMD59	Yes	[31:1] stuff bits, [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A "1" in the CRC option bit will turn the option on; a "0" will turn it off.
CMD60-CMD63	No	---	---	---	---

3.1.4.2.12– Δομή απαντήσεων

Υπάρχουν τρεις μορφές απαντήσεων στην SPI λειτουργία.

R1 μορφή: Αυτή η μορφή είναι η συνηθέστερη καθώς χρησιμοποιείται σε όλες τις εντολές πλην αυτών που ελέγχουν τη κατάσταση της κάρτας. Είναι μήκους ενός byte, μεταδίδεται με το MSB πρώτα ,το οποίο είναι πάντα λογικό 0 και τα υπόλοιπα bits είναι ενδείξεις λάθους (λογικό 1 σημαίνει λάθος)

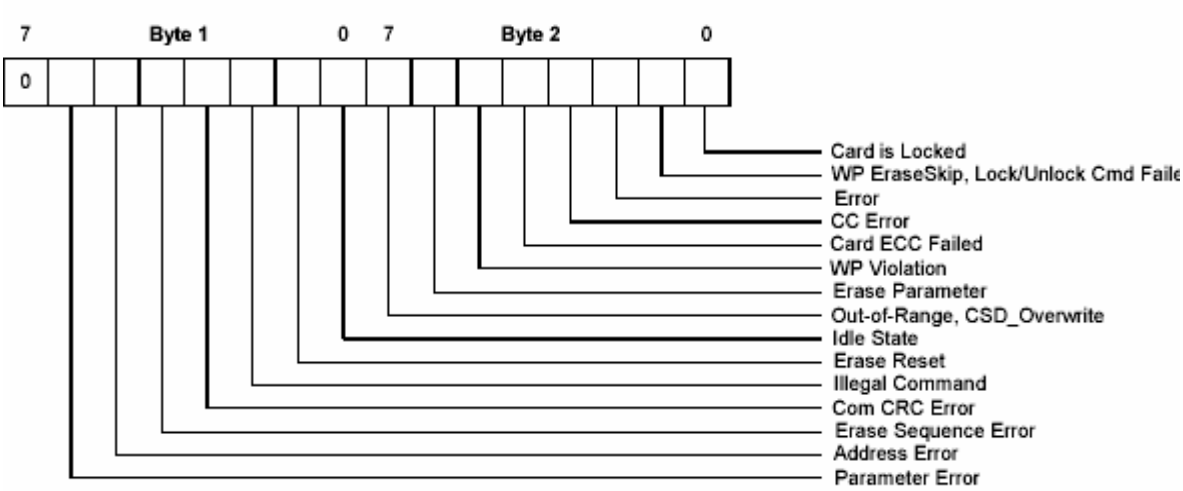


Εικόνα 3.16 : Μορφή απάντησης R1

Πίνακας 3.5 : Σημασία bits απάντησης R1

Error Indication	Definition
Idle State	The card is in an idle state and running initializing process.
Erase reset	An erase sequence was cleared before execution because an out-of-erase sequence command was received.
Illegal command	An illegal command code was detected.
Communication CRC error	The CRC check of the last command failed.
Erase sequence error	An error in the sequence of erase commands occurred.
Address error	A misaligned address that did not match the block length was used in the command.
Parameter error	The command's argument (e.g., address, block length) was out of the allowed range for this card.

R1b μορφή: Είναι πανομοιότυπη με την R1 μορφή με την προαιρετική προσθήκη σημαίας για τον έλεγχο της διαθεσιμότητας της κάρτας.
R2 μορφή: Η απάντηση αυτή δίνεται από την κάρτα αν ζητηθεί η κατάσταση της και έχει μήκος 2 bytes.

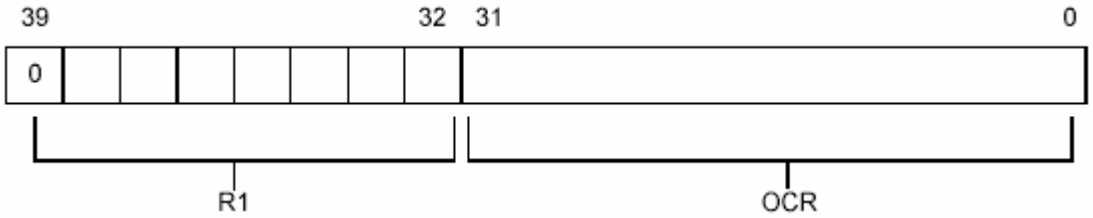


Εικόνα 3.17 : Μορφή απάντησης R2

Πίνακας 3.6 : Σημασία bits απάντησης R2

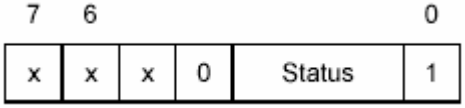
Error Indication	Definition
Out of range/CSD overwrite	This status bit has two functions. It is set if the command argument was out of its valid range, or if the host is trying to change the ROM section or reverse the copy bit (set as original) or permanent WP bit (un-protect) of the CSD register.
Erase parameter	An invalid selection, sectors, or groups for erase.
Write protect violation	The command tried to write to a write-protected block.
Card ECC failed	The card's internal ECC was applied but failed to correct the data.
CC error	Internal card controller error.
Error	A general or an unknown error occurred during the operation.
Write protect erase skip	This status bit has two functions. It is set when the host attempts to erase a write-protected sector or if a sequence or password error occurred during a card lock/unlock operation.
Card is locked	This bit is set when the user locks the card. It is reset when it is unlocked.

R3 μορφή: Η κάρτα στέλνει αυτή την απάντηση όταν της ζητηθεί το περιεχόμενο του OCR. Έχει μέγεθος 5 bytes από τα οποία το πρώτο είναι ίδιας μορφής με την R1 και τα υπόλοιπα είναι το περιεχόμενο του OCR.



Εικόνα 3.18 : Μορφή απάντησης R3

Μορφή απάντησης για δεδομένα: Κάθε block που γράφεται στην κάρτα τσεκάρεται από μία 1 byte μήκους απάντηση που έχει την ακόλουθη μορφή.



Εικόνα 3.19 : Μορφή απάντησης για δεδομένα

Το νόημα των STATUS bits είναι το ακόλουθο.

- 010: Τα δεδομένα έγιναν δεκτά
- 101: Τα δεδομένα απορρίφθηκαν λόγω ενός CRC λάθους
- 110: Τα δεδομένα απορρίφθηκαν λόγω ενός λάθους κατά την εγγραφή

3.1.4.2.13– Data tokens

Οι εντολές εγγραφής και ανάγνωσης συνοδεύονται από μεταφορά δεδομένων. Αυτές οι μεταφορές γίνονται πάντα μέσω data tokens. Αυτά μεταδίδονται πάντα με το MSB πρώτα, έχουν μήκος από 4 έως (N+3) bytes και έχουν την ακόλουθη μορφή:

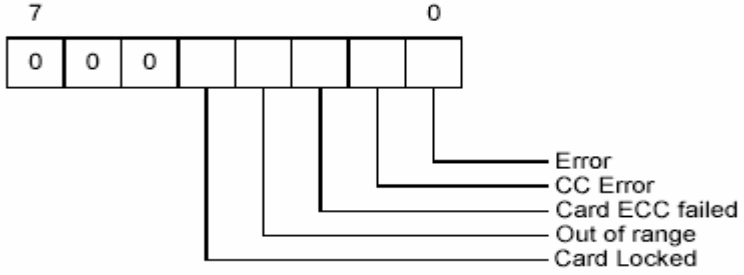
- Πρώτο byte: Αρχή block
- Δύο έως N+1: Δεδομένα
- Δύο τελευταία bytes: CRC

Πίνακας 3.7 : Αρχή block

Token Type	Transaction Type	7	Bit Position					0
Start Block	Single Block Read	1	1	1	1	1	1	1	0
Start Block	Multiple Block Read	1	1	1	1	1	1	1	0
Start Block	Single Block Write	1	1	1	1	1	1	1	0
Start Block	Multiple Block Write	1	1	1	1	1	1	0	0
Stop Tran	Multiple Block Write	1	1	1	1	1	1	0	1

3.1.4.2.14 – Data Error Token

Όταν συμβαίνει κάποιο λάθος η κάρτα απαντά με ένα Data Error Token το οποίο έχει την ακόλουθη μορφή:

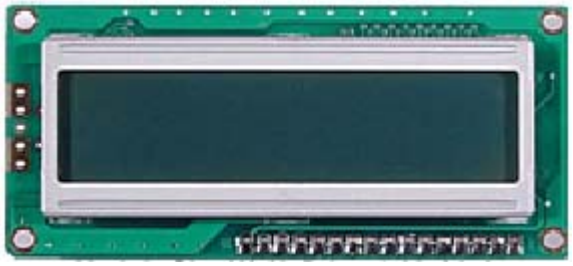


Εικόνα 3.20 : Μορφή error token

Οι χρονισμοί του καναλιού και ο τρόπος που χρησιμοποιήθηκε η κάρτα αναφέρονται αναλυτικά σε επόμενο κεφάλαιο που περιγράφεται το πρόγραμμα που συντάχθηκε και χρησιμοποιήθηκε για την οδήγηση της κάρτας.

3.2 – Οθόνη Χαρακτήρων (LCD Character)

Η οθόνη που χρησιμοποιήθηκε για την κατασκευή του MP3 αποκωδικοποιητή είναι η SSC2B16, μία οθόνη της εταιρείας SDEC , βασισμένη στον μικροϋπολογιστή HD44780 διαστάσεων 2x16 (δηλαδή έχει 2 γραμμές οι οποίες χωράνε μέχρι 16 χαρακτήρες η κάθε μία).



Εικόνα 3.21 : Οθόνη SSC2B16 2 x 16 βασισμένη στον μικροελεγκτή HD44780

3.2.1 - Γενική Περιγραφή

Η οθόνη λαμβάνει 8-bit κωδικούς από έναν μικροϋπολογιστή ή έναν μικροελεγκτή μεταφέρει αυτό τον κωδικό στην DDRAM (80 bytes Data Display RAM για την αποθήκευση 80 χαρακτήρων), η οποία μετατρέπει τον κωδικό στον αντίστοιχο χαρακτήρα για την απεικόνιση του στην οθόνη σαν πίνακα κουκίδων 5 x 7. Ακόμα η οθόνη περιέχει 160 bytes ROM η οποία παρέχει τις γραφικές αναπαραστάσεις των χαρακτήρων. Επίσης υποστηρίζει την δημιουργία μέχρι 8 νέων χαρακτήρων, καθώς διαθέτει 64 bytes RAM, για τις απαιτήσεις του κάθε προγράμματος.

Για την απεικόνιση ενός χαρακτήρα σε συγκεκριμένη θέση ο ελεγκτής θέτει την επόμενη θέση, στέλνοντας κατάλληλη εντολή στην οθόνη, και έπειτα στέλνει τον κωδικό του χαρακτήρα και αυτός απεικονίζεται στην οθόνη στην καθορισμένη θέση. Η οθόνη μπορεί μόνη της να αυξάνει ή να μειώνει την θέση του κέρσορα με αποτέλεσμα η αναγραφή ενός αλφαριθμητικού να αποτελεί υπόθεση απλώς σωστής σειριακής αποστολής έγκυρων κωδικών. Ακόμα η αναγραφή μπορεί να γίνει από τα δεξιά στα αριστερά και αντίθετα.

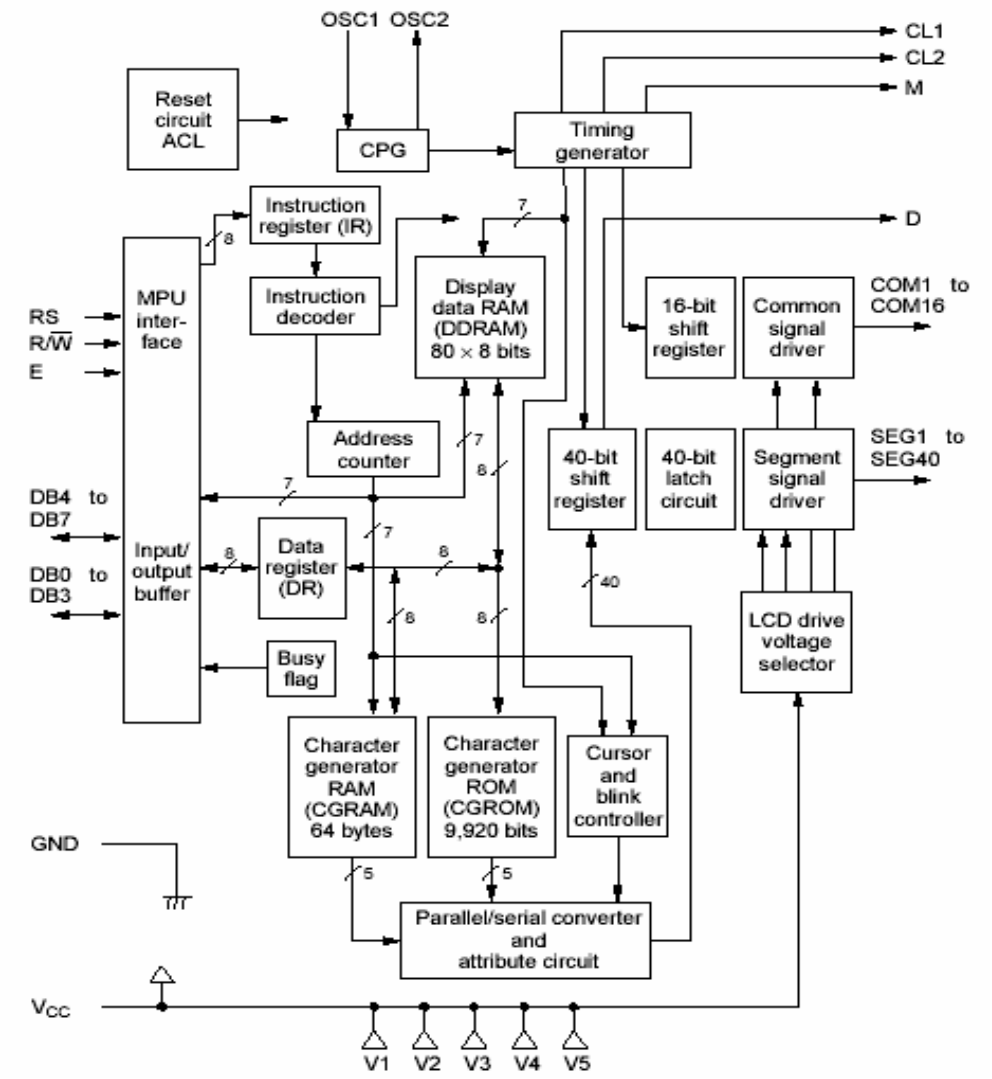
Η οθόνη μπορεί να χειριστεί είτε σε 8-bit λειτουργία, είτε σε 4-bit λειτουργία κάτι που ελαχιστοποιεί τα σήματα που απαιτούνται για την σωστή λειτουργία της. Τέλος είναι χαμηλής κατανάλωσης κάτι που την κάνει ιδανική για εφαρμογές σε αυτόνομα, φορητά συστήματα.

3.2.2 – Χαρακτηριστικά - Αρχιτεκτονική

Η οθόνη έχει συνοπτικά τα παρακάτω χαρακτηριστικά:

- Λειτουργία και σε 8-bit και 4-bit.
- Εσωτερική DRAM
- 80 χαρακτήρες αποθηκευμένοι στην DRAM
- Μνήμη ROM για την δημιουργία του χαρακτήρα στην οθόνη
- 160 διαφορετικά γραφικά για την απεικόνιση ων 5 x 7 χαρακτήρων
- Επιπρόσθετη μνήμη RAM για την δημιουργία μέχρι 8 διαφορετικών 5 x 7, καθορισμένων από τον χρήστη χαρακτήρων.
- Εσωτερικό κύκλωμα αρχικοποίησης της μετά από κάθε τροφοδότησή της.
- Πλήθος εντολών όπως: καθαρισμός της οθόνης, ολίσθησή της δεξιά αριστερά, τοποθέτηση του κέρσορα και πολλές άλλες.
- Εσωτερικό ρολόι.

Όπως αναφέρθηκε και προηγουμένως η οθόνη είναι βασισμένη στον μικροελεγκτή HD44780. Η αρχιτεκτονική του φαίνεται στην παρακάτω εικόνα:



Εικόνα 3.22 : Αρχιτεκτονική του μικροελεγκτή HD44780

Η οθόνη χρησιμοποιεί 3 σήματα για τον έλεγχο της και 8 γραμμές για την μεταφορά των δεδομένων. Τα τρία σήματα ελέγχου είναι τα E, RS, R/W ενώ τα σήματα δεδομένων οι γραμμές DB0-DB7. Αυτά φαίνονται στον παρακάτω πίνακα.

Πίνακας 3.8 : Pin configuration της οθόνης

NO	SYM	LEVEL	FUNCTION	NO	SYM	LEVEL	FUNCTION
1	VSS	-	0V	9	DB2	H/L	DATA BIT2
2	VDD	-	5V	10	DB3	H/L	DATA BIT3
3	VO	-	CONTRAST ADJ	11	DB4	H/L	DATA BIT4
4	RS	H/L	REGISTER SELECT	12	DB5	H/L	DATA BIT5
5	R/W	H/L	READ/WRITE	13	DB6	H/L	DATA BIT6
6	E	H.H->L	ENABLE SIGNAL	14	DB7	H/L	DATA BIT7
7	DB0	H/L	DATA BIT0	15	A(+)	5V	BACKLIGHT
8	DB1	H/L	DATA BIT1	16	K(-)	0V	BACKLIGHT

Ο μικροελεγκτής έχει 2 καταχωρητές μεγέθους 8-bit , τον καταχωρητή εντολών IR (Instruction Register) και τον καταχωρητή δεδομένων DR (Data Register). Ο καταχωρητής δεδομένων περιέχει τους κωδικούς των εντολών και πληροφορίες για τις μνήμες της οθόνης. Είναι προσβάσιμος μόνο για εγγραφή από τον μικροελεγκτή.

Ο καταχωρητής δεδομένων χρησιμοποιείται για προσωρινή καταχώρηση δεδομένων κατά την διάρκεια της επικοινωνίας της οθόνης με τον μικροϋπολογιστή. Έπειτα τα δεδομένα μεταφέρονται στην κατάλληλη μνήμη για αποκωδικοποίηση τους και μεταφορά στην οθόνη. Ακόμα χρησιμοποιείται όταν ο μικροϋπολογιστής θέλει να διαβάσει δεδομένα από την οθόνη. Η επιλογή του καταχωρητή γίνεται με την χρήση του σήματος RS (Register Select).

Πίνακας 3.9 : Χρήση των σημάτων RS – R/W

RS	R/W	Λειτουργία
0	**0**	**Εγγραφή του IR και εκτέλεση εσωτερικής λειτουργίας**
0	**1**	**Ανάγνωση της BF και της διεύθυνσης**
1	**0**	**Εγγραφή του DR και εκτέλεση εσωτερικής λειτουργίας**
1	**1**	**Ανάγνωση του DR και εκτέλεση εσωτερικής λειτουργίας**

Η οθόνη χρησιμοποιεί μία σημαία για να υποδείξει ότι ακόμα δουλεύει σε εσωτερική λειτουργία και δεν είναι έτοιμη να δεχτεί καινούργιες εντολές. Η σημαία αυτή, που από αυτό το σημείο και πέρα θα αποκαλείται BF (Busy Flag), μπορεί να αναγνωστεί από το σήμα δεδομένων DB7 όταν το σήμα RS είναι λογικά χαμηλό και το σήμα R/W λογικά ψηλό. Η οθόνη δεικνύει ότι είναι απασχολημένη σε εσωτερική λειτουργία όταν αυτή η σημαία είναι λογικό 1 και μηδενίζει την σημαία όταν είναι έτοιμη να δεχτεί νέες εντολές.

Η οθόνη έχει ένα μετρητή διευθύνσεων AC (Address Counter) ο οποίος δημιουργεί την διεύθυνση για την DDRAM και για την RAM που περιέχει τους χαρακτήρες του χρήστη.

Η 80 x 8 bit RAM της οθόνης περιέχει 80 8-bit κωδικούς χαρακτήρων. Η αχρησιμοποίητη περιοχή μνήμης μπορεί να χρησιμοποιηθεί για γενική χρήση από τον μικροϋπολογιστή. Οι διευθύνσεις της DDRAM αντιστοιχούν στις θέσεις της οθόνης (για οθόνη 2 x 16 που χρησιμοποιήθηκε) όπως φαίνεται στο παρακάτω σχήμα:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	← DISPLAY POSITION
FIRST LINE	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	DD RAM ADDRESS
SECOND LINE	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	

Εικόνα 3.23 : Αντιστοιχία θέσης και διεύθυνσης DDRAM

Για κάθε γραμμή ανατίθονται 40 θέσεις μνήμης. Έτσι για οθόνες με μήκος μικρότερο από 40 χαρακτήρες η πρώτη διεύθυνση της δεύτερης γραμμής δεν είναι η φυσικά επόμενη της τελευταίας της πρώτης γραμμής. Όταν γίνεται ολίσθηση της οθόνης το αποτέλεσμα στις διευθύνσεις είναι όπως το παρακάτω (για οθόνη 2 x 40 χαρακτήρων):

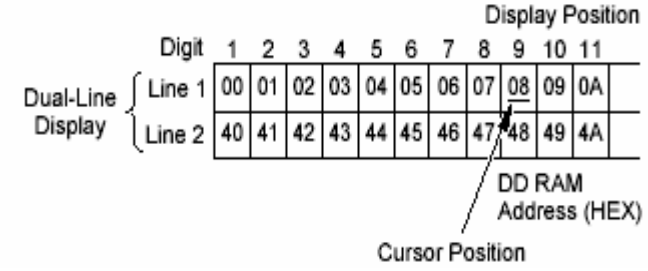
Left Shift	01_H	02_H	03_H	04_H	05_H	06_H	07_H	08_H	09_H	. . .	27_H	00_H
	41_H	42_H	43_H	44_H	45_H	46_H	47_H	48_H	49_H	. . .	67_H	40_H

Right Shift	27_H	00_H	01_H	02_H	03_H	04_H	05_H	06_H	07_H	. . .	25_H	26_H
	67_H	40_H	41_H	42_H	43_H	44_H	45_H	46_H	47_H	. . .	65_H	66_H

Εικόνα 3.24 : Αποτέλεσμα αριστερής και δεξιάς ολίσθησης αντίστοιχα

Η οθόνη έχει μνήμη ROM η οποία περιέχει την γραφική αναπαράσταση 160 κωδικών χαρακτήρων για την απεικόνιση τους στην οθόνη. Η γραφική αναπαράσταση έχει μέγεθος 5 x 7 (δηλαδή είναι πίνακας με 35 leds). Ο κωδικός των χαρακτήρων μαζί με την γραφική τους αναπαράσταση φαίνεται σε επόμενο πίνακα. Εναλλακτικά μπορούν να δημιουργηθούν καινούργιοι τέτοιοι πίνακες από τον χρήστη στην RAM που υπάρχει για τον χρήστη. Αξίζει να σημειωθεί ότι οι κωδικοί των χαρακτήρων είναι στην πραγματικότητα οι ASCII κωδικοί των χαρακτήρων, κάτι που μειώνει τον προγραμματιστικό χρόνο που χρειάζεται.

Η οθόνη έχει ακόμα εσωτερικό ρολόι για την δημιουργία των απαραίτητων χρονισμών για τις εσωτερικές λειτουργίες της. Ακόμα υπάρχει εσωτερικός μηχανισμός για την διαχείριση του κέρσορα (εμφάνιση ή απόκρυψη στην θέση που δείχνει ο AC) και την απόκρυψη οποιουδήποτε χαρακτήρα στην οθόνη.



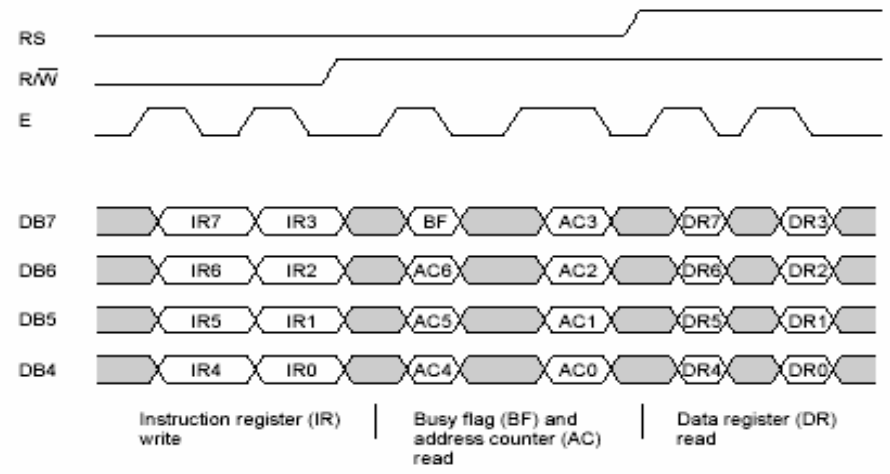
Εικόνα 3.25 : Παράδειγμα εμφάνισης του κέρσορα

Πίνακας 3.10 : Γραφική αναπαράσταση κωδικών

HIGH-ORDER 4 BIT / LOW-ORDER 4 BIT	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)												
xxxx0001	(2)												
xxxx0010	(3)												
xxxx0011	(4)												
xxxx0100	(5)												
xxx0101	(6)												
xxx0110	(7)												
xxxx0111	(8)												
xxxx1000	(1)												
xxxx1001	(2)												
xxxx1010	(3)												
xxxx1011	(4)												
xxxx1100	(5)												
xxxx1101	(6)												
xxxx1110	(7)												
xxxx1111	(8)												

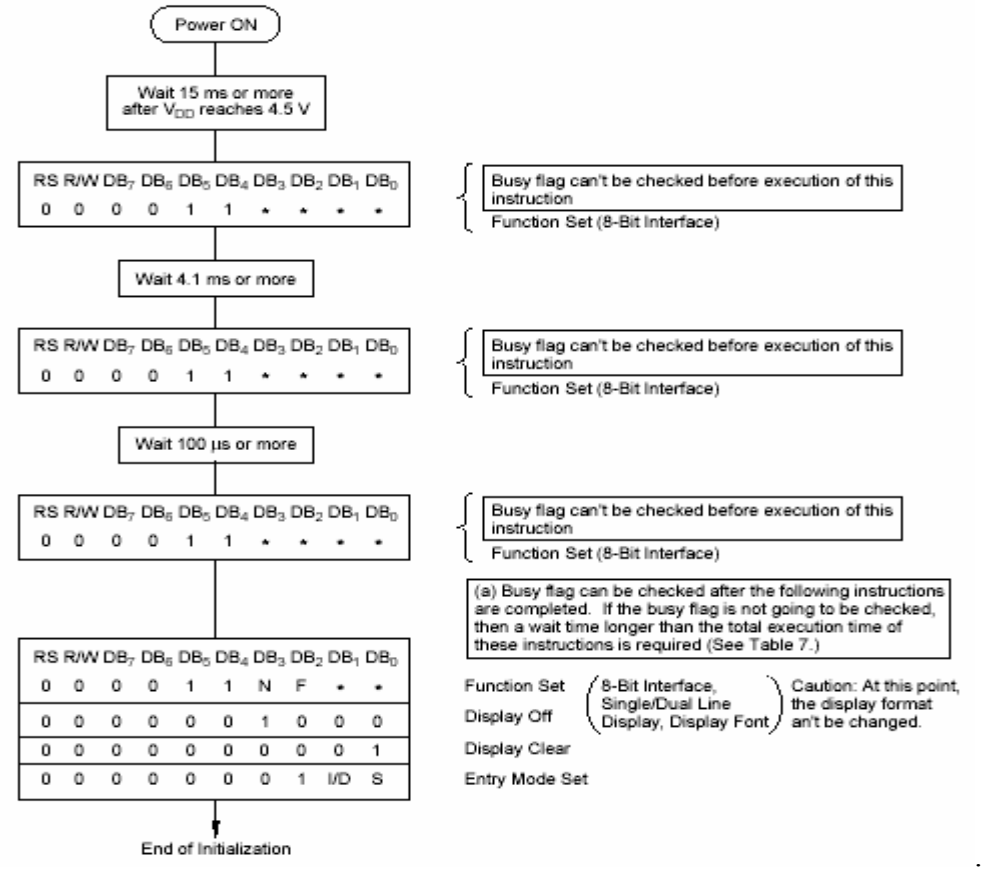
3.2.3 – Interface του μικροελεγκτή HD44780 -Αρχικοποίηση

Ο μικροελεγκτής HD44780 που περιέχει η οθόνη λειτουργεί σε 4-bit ή σε 8-bit λειτουργία. Στην λειτουργία σε 8-bit τα δεδομένα μεταφέρονται σε μία μεταφορά από τις γραμμές DB0-DB7. Αντίθετα στην 4-bit λειτουργία χρησιμοποιούνται μόνο οι γραμμές DB4-DB7 και τα δεδομένα μεταφέρονται σε 2 χρόνους με τα μεγαλύτερης αξίας bits να μεταφέρονται πρώτα. Σε αυτή την λειτουργία τα υπόλοιπα pins (DB0-DB3) μπορούν να συνδεθούν στην γείωση απευθείας.

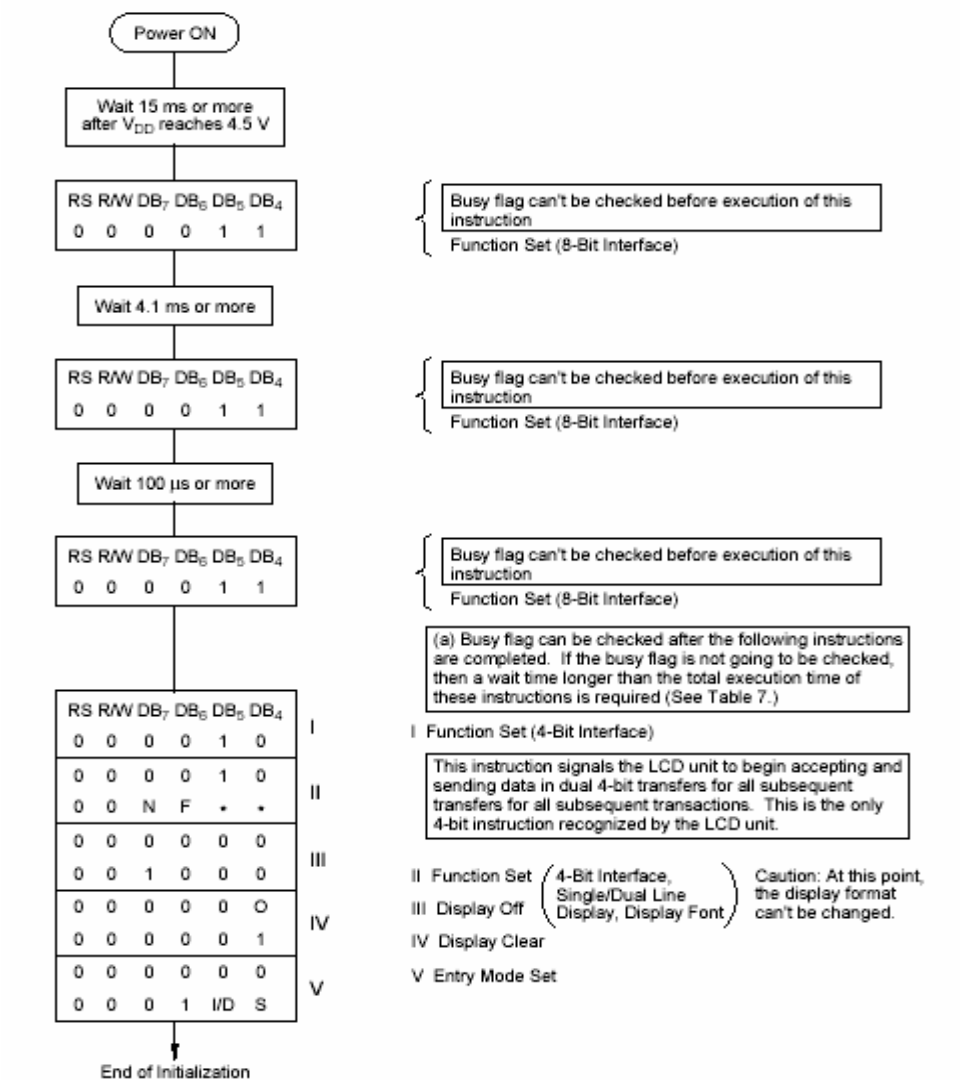


Εικόνα 3.26 : Μεταφορά δεδομένων σε 4-bit λειτουργία

Η οθόνη έχει εσωτερικό κύκλωμα το οποίο αρχικοποιεί την οθόνη μετά την τροφοδότηση της με ρεύμα. Μερικές φορές η οθόνη δεν αρχικοποιείται από το εσωτερικό κύκλωμα και πρέπει να αρχικοποιηθεί μέσω εντολών. Αυτή η αρχικοποίηση περιγράφεται αναλυτικά από το επόμενο σχήμα.



Εικόνα 3.27 : Αρχικοποίηση μέσω εντολών σε 8-bit λειτουργία



Εικόνα 3.28 : Αρχικοποίηση μέσω εντολών σε 4-bit λειτουργία

Η διαδικασία αρχικοποίησης περιγράφεται αναλυτικά κατά την περιγραφή του προγράμματος.

3.2.4 – Εντολές

Η οθόνη υποστηρίζει μία μεγάλη ποικιλία εντολών κάτι που κάνει την οδήγησή της πολύ πιο εύκολη υπόθεση. Αυτές μπορούν να καταταχθούν σε 4 γενικές κατηγορίες:

1. Εντολές για την συμπεριφορά της οθόνης
2. Εντολές για την διευθυσιοδότηση της εσωτερικής RAM
3. Εντολές για την μεταφορά δεδομένων από και προς τη RAM
4. Άλλες εντολές

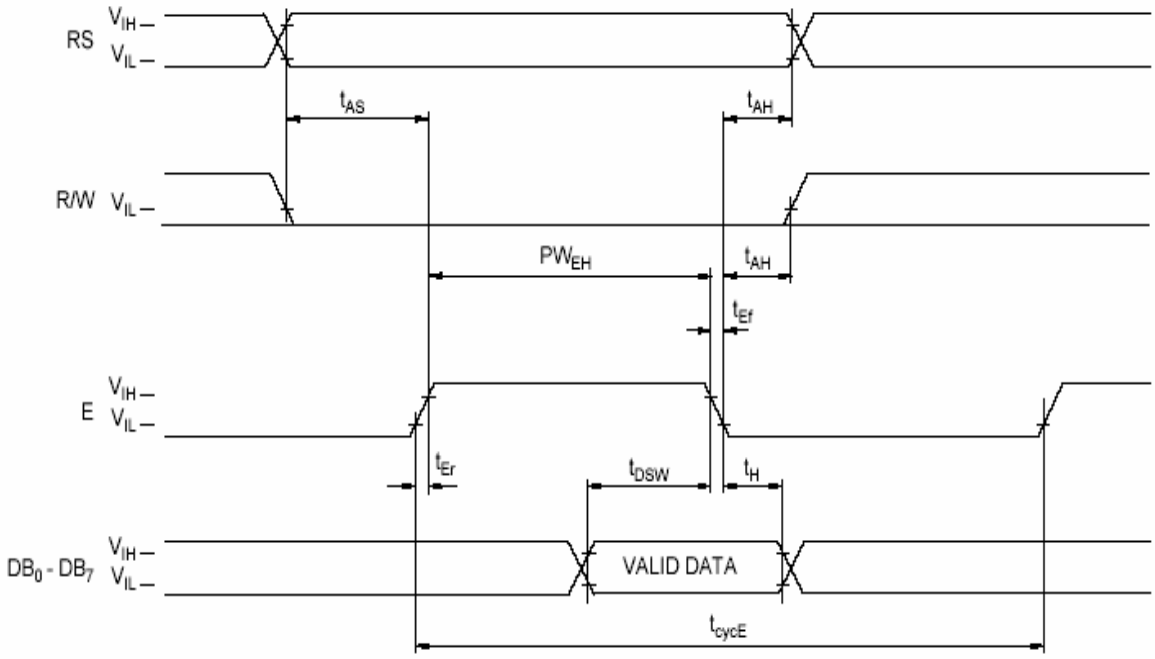
Η λίστα των εντολών φαίνεται στον επόμενο πίνακα:

Πίνακας 3.11 : Εντολές που υποστηρίζει η οθόνη

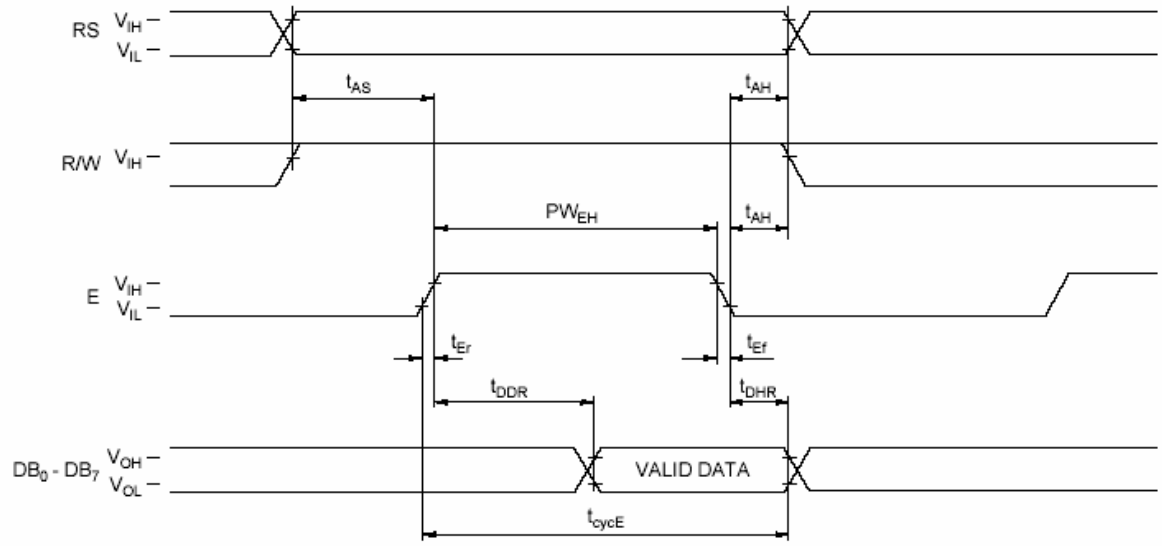
Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears the display and returns the cursor to the home position (address 0).	82μs~1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (address 0). Also returns a shifted display to the home position. DD RAM contents remain unchanged.	40μs~1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and enables/disables the display.	40μs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Turns the display ON/OFF (D), or the cursor ON/OFF (C), and blink of the character at the cursor position (B).	40μs
Cursor & Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing the DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N$	F	*	#	Sets the data width (DL), the number of lines in the display (L), and the character font (F).	40μs
Set CG RAM Address	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data can be read or altered after making this setting.	40μs
Set DD RAM Address	0	0	1	A_{DD}							Sets the DD RAM address. Data may be written or read after making this setting.	40μs
Read Busy Flag & Address	0	1	BF	AC							Reads the BUSY flag (BF) indicating that an internal operation is being performed and reads the address counter contents.	1μs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	46μs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	46μs
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift. S/C= 1: Display shift S/C = 0: cursor move R/L= 1: Shift to the right. R/L= 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5x10 dots F = 0: 5 x 7 dots BF = 1: Busy BF = 0: Can accept data *# Set to 1 on 24x4 modules* *$ With KS0072 is Address Mode.*											DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM Address A_{DD}: DD RAM Address Corresponds to cursor address. AC: Address counter Used for both DD and CG RAM address.	Execution times are typical. If transfers are timed by software and the busy flag is not used, add 10% to the above times.

3.2.5 – Χρονικά χαρακτηριστικά

Ένα από τα πιο σημαντικά χαρακτηριστικά της οθόνης είναι το πότε δειγματοληπτούνται τα δεδομένα. Αυτό γίνεται στην μετάβαση της γραμμής Enable από 1 σε 0. Κατά την διάρκεια της μεταφοράς δεδομένων πρέπει να τηρούνται κάποιοι χρονισμοί που φαίνονται στο παρακάτω σχήμα:



Εικόνα 3.29 : Χρονισμοί μεταφοράς δεδομένων από MCU σε οθόνη



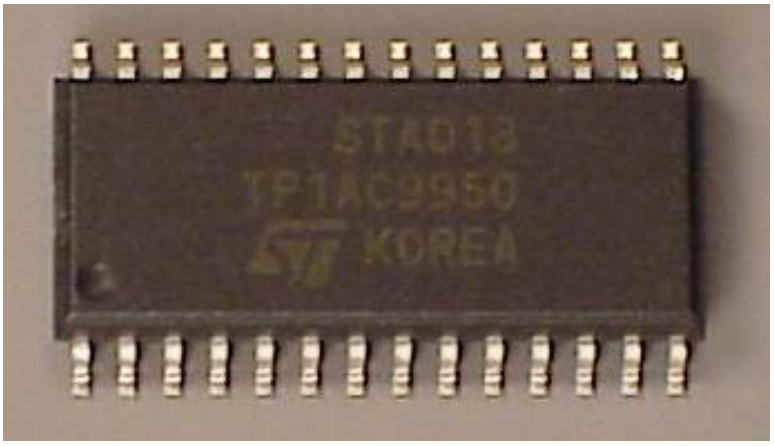
Εικόνα 3.30 : Χρονισμοί μεταφοράς δεδομένων από οθόνη σε MCU

Πίνακας 3.12 : Ελάχιστοι χρόνοι που υποστηρίζει η οθόνη

PARAMETER		SYMBOL	VALUE		UNIT
			MIN.	MAX.	
Enable Cycle Time		t_{cycE}	1000	—	ns
Enable Pulse Width	"High" Level	PW_{EH}	450	—	ns
Enable Rise/Fall Time		t_{Er}, t_{Ef}	—	25	ns
Setup Time	RS, R/W-E	t_{AS}	140	—	ns
Address Hold Time		t_{AH}	10	—	ns
Data Setup Time		t_{DSW}	195	—	ns
Data Hold Time		t_H	10	—	ns

3.3 – Ο αποκωδικοποιητής των mp3 – STA013

Για την κατασκευή του mp3 αποκωδικοποιητή χρησιμοποιήθηκε για την αποκωδικοποίηση των mp3 τραγουδιών το ολοκληρωμένο STA013 της ST Microelectronics.



Εικόνα 3.31 : Ο αποκωδικοποιητής των mp3 STA013

Το STA013 αναγνωρίζει μόνο του το bitrate των τραγουδιών και τη συχνότητα δειγματοληψίας των mp3 και δημιουργεί όλα τα απαραίτητα σήματα για έναν ψηφιακό σε αναλογικό μετατροπέα. Το μόνο που έχει να κάνει ο προγραμματιστής είναι να φροντίσει ώστε να τροφοδοτείται σωστά το STA013 για να μην μείνει ποτέ χωρίς δεδομένα για αποκωδικοποίηση.

Το STA013 απαιτεί για τον έλεγχο του το I2C πρωτόκολλο. Είναι αδύνατο να χρησιμοποιηθεί το ολοκληρωμένο χωρίς την χρησιμοποίηση του I2C. Παρακάτω αναλύεται το I2C της Phillips.

3.3.1 – Το interface I^2C της Phillips

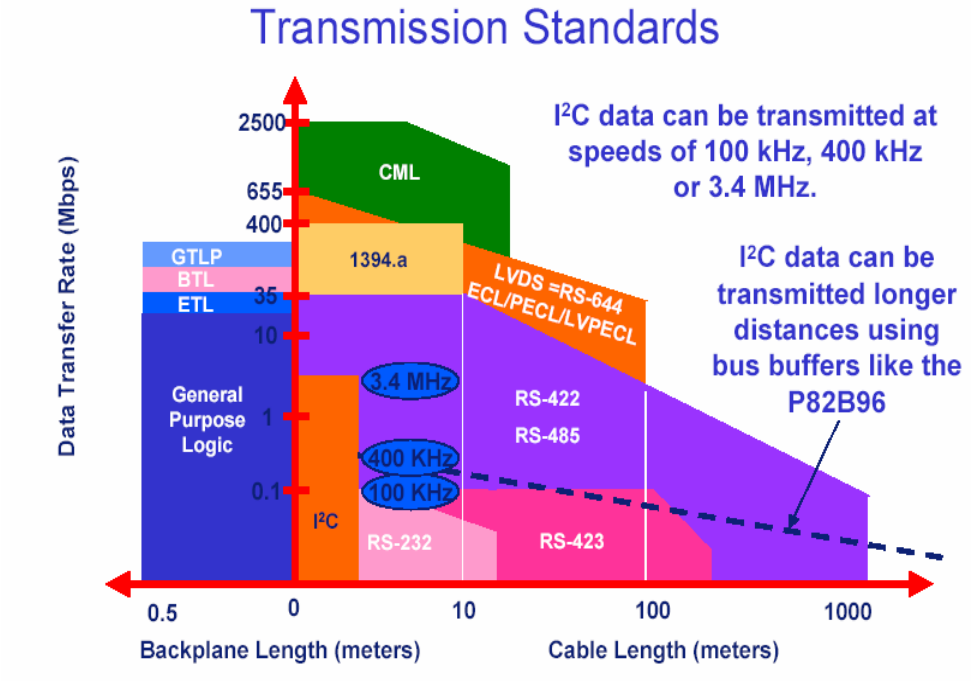
Το Inter IC bus, συνήθως γνωστό ως I²C bus, είναι ένα bus ελέγχου που παρέχει τη σύνδεση επικοινωνιών μεταξύ των ολοκληρωμένων κυκλωμάτων σε ένα σύστημα. Αναπτυγμένο από την Philips στις αρχές της δεκαετίας του '80, αυτό το απλό two-wire bus με ένα λογισμικά-καθορισμένο πρωτόκολλο έχει εξελιχθεί για να γίνει το de facto παγκόσμια πρότυπο για τον έλεγχο συστημάτων, βρίσκοντας εφαρμογές του σε όλα, από αισθητήρες θερμοκρασίας και μεταφραστές επιπέδων τάσης σε EEPROMs, γενικής χρήσης μετατροπείς I/O, A/D και D/A, CODECs, και μικροεπεξεργαστές όλων των ειδών.

Υπάρχουν διάφοροι λόγοι για τους οποίους το I^2C -bus έχει διαρκήσει για περισσότερο από 20 έτη. Για να αρχίσουμε, το I^2C έχει συμβαδίσει με την απόδοση και παρέχει σήμερα τρία επίπεδα μεταφοράς δεδομένων: μέχρι 100 kbps στον Standard mode, μέχρι 400 kbps στον Fast mode, και μέχρι 3,4 Mbps στο High-Speed mode. Τα πρόσφατα εισαχθέντα hubs, οι επαναλήπτες bus, οι αμφίδρομοι διακόπτες και οι πολυδιαυλωτές έχουν αυξήσει τον αριθμό συσκευών που το bus μπορεί να υποστηρίξει, επεκτείνοντας την ικανότητα μέγιστης χωρητικότητας πέρα από το αρχικό μέγιστό του 400 pF. Επίσης, η λογισμικά-ελεγχόμενη ανίχνευση σύγκρουσης και η διαιτησία αποτρέπουν τη αποτυχία αποστολής δεδομένων και εξασφαλίζουν αξιόπιστη απόδοση, ακόμη και στα σύνθετα συστήματα.

Πέρα από την απόδοση, εν τούτοις, υπάρχει ευκολία στη χρήση. Δύο απλές γραμμές συνδέουν όλο τα ICs σε ένα σύστημα. Οποιαδήποτε συσκευή I²C μπορεί να συνδεθεί με ένα κοινό I²C - bus, και οποιαδήποτε κύρια συσκευή μπορεί να ανταλλάξει τις πληροφορίες με

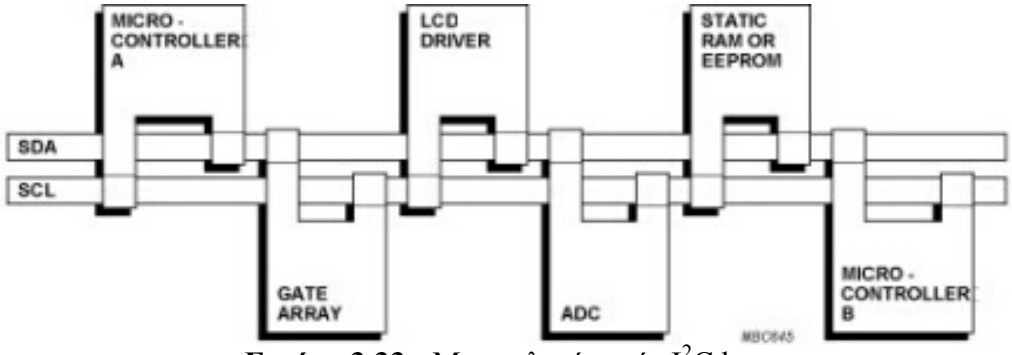
οποιαδήποτε συσκευή σκλάβο. Το λογισμικά-ελεγχόμενο σχέδιο διευθυνσιοδότησης εξαλείφει την ανάγκη για υλικό αποκωδικοποίησης διευθύνσεων, και έτσι δεν προκύπτει καμία ανάγκη να σχεδιαστεί και να διορθωθεί η εξωτερική λογική ελέγχου επειδή παρέχεται ήδη από το πρωτόκολλο I²C.

Οι σχεδιαστές μπορούν να κινηθούν γρήγορα από το μπλοκ διάγραμμα προς το τελικό υλικό, απλά συνδέοντας νέες συσκευές και λειτουργίες σε ένα υπάρχον bus. Το I²C -bus εξοικονομεί χώρο και χαμηλώνει το γενικό κόστος. Η δομή δύο-γραμμών σημαίνει λιγότερες γραμμές, έτσι το PCB μπορεί να είναι πολύ μικρότερο. Η απασφαλμάτωση είναι ευκολότερη επίσης, δεδομένου ότι υπάρχουν λιγότερες γραμμές και λιγότερες πηγές πληροφοριών να ελεγχθούν. Καθώς το σύστημα εξελίσσεται συνεχώς, οι συσκευές I²C μπορούν εύκολα να προστεθούν ή να αφαιρεθούν χωρίς δημιουργία προβλημάτων στο υπόλοιπο του συστήματος.



Εικόνα 3.32 : Περιοχές λειτουργίας διάφορων interfaces

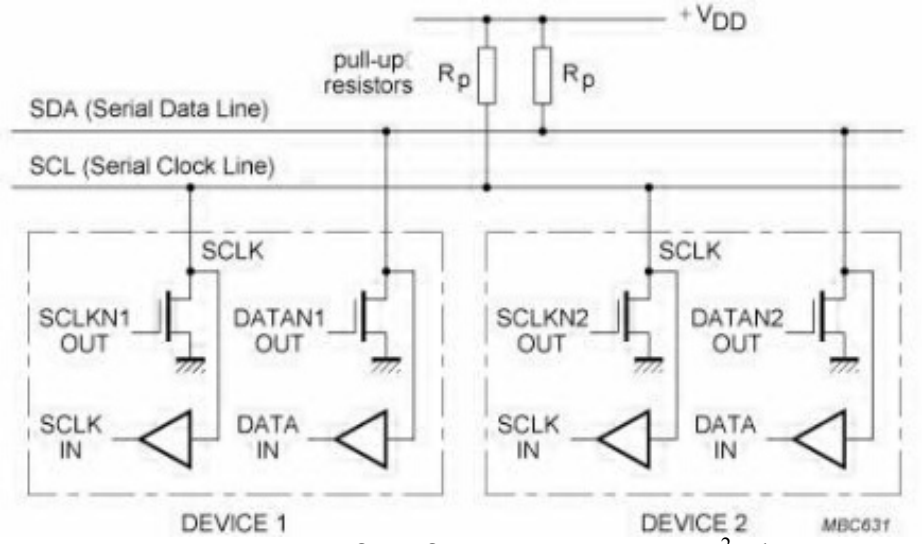
3.3.1.1 – Πως λειτουργεί το I²C της Phillips



Εικόνα 3.33 : Μορφολογία ενός I²C bus

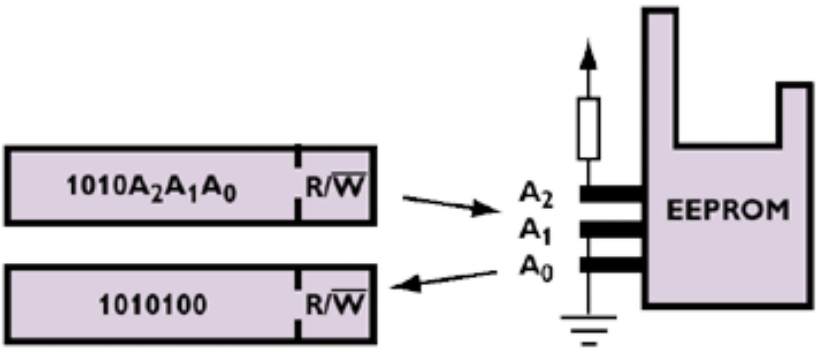
Οποιαδήποτε συσκευή I²C μπορεί να συνδεθεί με ένα I²C -bus και κάθε συσκευή μπορεί να μιλήσει με οποιοδήποτε master, που διακινεί τις πληροφορίες. Πρέπει να είναι τουλάχιστον ένας master (π.χ., μικροελεγκτής ή DSP) στο bus αλλά μπορούν να υπάρξουν

περισσότεροι του ενός, με όλους τους master να έχουν την ίση προτεραιότητα. Οι συσκευές μπορούν να προστεθούν εύκολα και να αφαιρεθούν από το I²C -bus.



Εικόνα 3.34 : Σύνδεση δύο συσκευών σε ένα I²C bus

Η συνολική χωρητικότητα του bus πρέπει να είναι λιγότερο από 400 pF (π.χ., περίπου 20-30 συσκευές ή 10 μ γραμμής) για τον σεβασμό στις χρονικές απαιτήσεις ανόδου και πτώσης. Κάθε συσκευή πρέπει να είναι σε θέση να οδηγήσει μέχρι 3 mA για ένα χαμηλό επίπεδο λογικής 0,4 mA σε ένα ανοικτό bus αγωγών με την pull-up στη σειρά να έχει τιμή από 2 K μέχρι 10 K. Οι αμφίδρομοι απομονωτές bus I²C είναι διαθέσιμοι για την απομόνωση στα διαφορετικά κομμάτια του bus για την αύξηση τόσο της συνολικής χωρητικότητας ή απόστασης του bus.

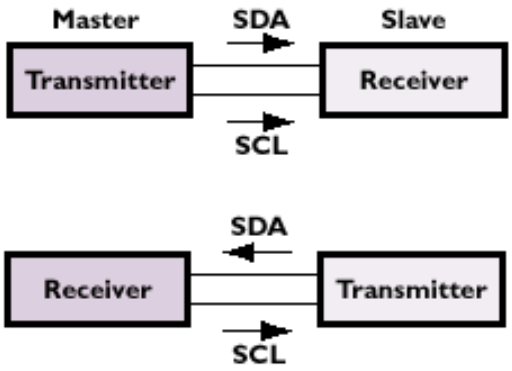


Εικόνα 3.35: Διευθυσιοδότηση σε ένα I²C bus

Κάθε συσκευή έχει μια μοναδική επτάμπιτη διεύθυνση I²C έτσι ώστε ο master να ξέρει συγκεκριμένα με ποιον επικοινωνεί. Χαρακτηριστικά τα τέσσερα σημαντικότερα bits καθορίζονται και ορίζονται σε συγκεκριμένες κατηγορίες συσκευών (π.χ. 1010 ορίζονται σε σειριακές EEPROMs). Τα τρία λιγότερο σημαντικά bits (A2, A1 και A0) είναι προγραμματίσημα μέσω των διευθύνσεων υλικού που επιτρέπουν μέχρι οκτώ διαφορετικούς συνδυασμούς διευθύνσεων I²C και επομένως επιτρέπουν μέχρι σε οκτώ από εκείνο τον τύπο συσκευής για να λειτουργήσουν στο ίδιο I²C -bus. Αυτές οι έξοδοι κρατιούνται υψηλές σε Vcc (1) ή χαμηλά gnd (0). Η επτάμπιτη διευθυσιοδότηση επιτρέπει

μέχρι 128 συσκευές στο ίδιο bus αλλά μερικές από αυτές τις διευθύνσεις είναι διατηρημένες για ειδικές εντολές έτσι ώστε το πρακτικό όριο να είναι περίπου 120.

3.3.1.2 – Ορολογία του I²C της Phillips

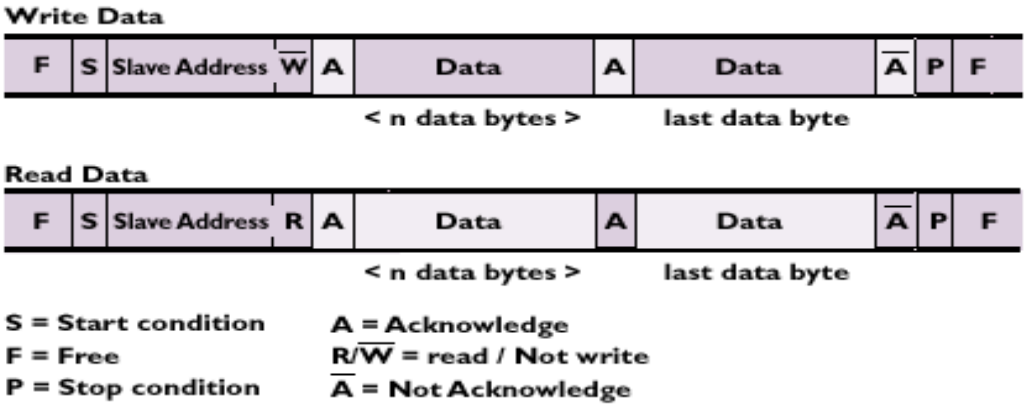


Εικόνα 3.36 : Ορολογία I²C bus

- **Transmitter** - η συσκευή που στέλνει τα στοιχεία στο bus. Μια συσκευή αποστολής σημάτων μπορεί είτε να είναι μια συσκευή που βάζει τα στοιχεία όσον αφορά το bus στη συμφωνία της (μια "master-συσκευή αποστολής σημάτων"), ή σε απάντηση σε ένα αίτημα από τον master (μια "slave-συσκευή αποστολής σημάτων").
- **Receiver** - η συσκευή που λαμβάνει τα στοιχεία από το bus. Ένας δέκτης μπορεί είτε να είναι μια συσκευή που λαμβάνει τα στοιχεία όσον αφορά το αίτημά της (ένας "master-δέκτης), ή σε απάντηση σε ένα αίτημα από τον master (ένας "slave-δέκτης).
- **Master** – Η συσκευή που αρχίζει μια μεταφορά (εντολή έναρξης), παράγει το σήμα ρολογιού (SCL) και ολοκληρώνει τη μεταφορά (εντολή τερματισμού). Ένας master μπορεί να είναι είτε μια συσκευή transmitter είτε μια receiver.
- **Slave** - η συσκευή που καλείται από τον master. Ένας σκλάβος μπορεί να είναι είτε transmitter είτε receiver.
- **Multi-Master** - η δυνατότητα για περισσότερους από έναν master να συνυπάρξουν στο bus συγχρόνως χωρίς απώλεια λόγω σύγκρουσης δεδομένων. Οι χαρακτηριστικές "bit - banged" εφαρμογές λογισμικού δεν είναι multi-master ικανές. Ο παράλληλος ελεγκτής στα I²C – bus παρέχει έναν εύκολο τρόπο να προστεθεί ένας multi-master ελεγκτής υλικού I²C σε DSPs και ASICs.
- **Arbitration** - η προσχεδιασμένη διαδικασία που εγκρίνει μόνο έναν master τη φορά να πάρει τον έλεγχο του bus.
- **Synchronization** - η προσχεδιασμένη διαδικασία που συγχρονίζει τα σήματα ρολογιών που παρέχονται από δύο ή περισσότερους master.
- **SDA** - γραμμή σημάτων δεδομένων (Serial DAta)
- **SCL** - γραμμή σημάτων ρολογιών (Serial CLock)

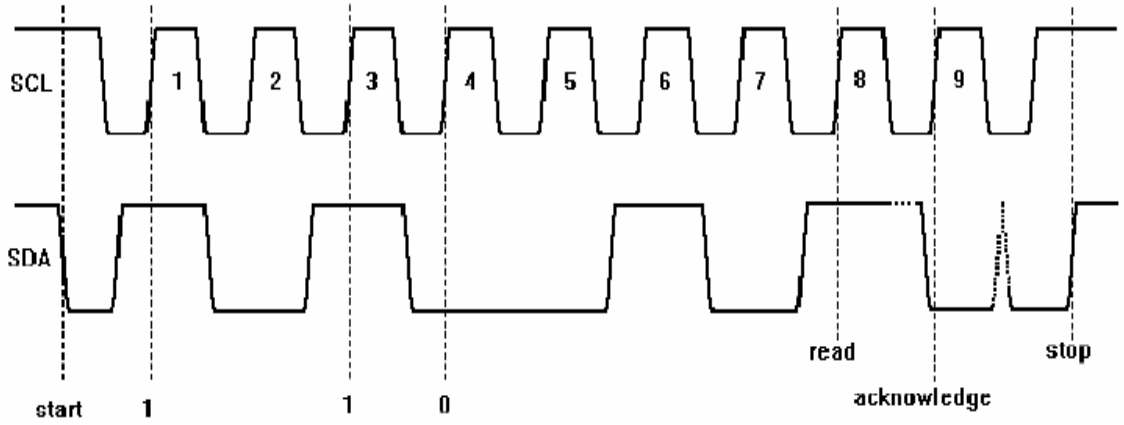
Η διεύθυνση I²C της στοχοθετημένης συσκευής στέλνεται στο 1ο byte και το σημαντικό κομμάτι αυτό δείχνει εάν ο master πρόκειται να στείλει (να γράψουν) ή να λάβει (να διαβάσει) στοιχεία από το δέκτη, αποκαλούμενο και slave. Κάθε ακολουθία μετάδοσης πρέπει να αρχίσει με τον όρο Start και να τελειώσει με τον όρο Stop ή τον όρο Restart. Εάν υπάρχουν δύο master στο ίδιο I²C - bus, υπάρχει μια διαδικασία διαιτησίας εάν και οι δύο προσπαθούν να πάρουν τον έλεγχο του bus συγχρόνως με να παραγάγουν την εντολή

έναρξης συγχρόνως. Όταν ένας master (π.χ., μικροελεγκτής) έχει τον έλεγχο του bus, κανένας άλλος master δεν μπορεί να πάρει τον έλεγχο έως ότου στείλει ο πρώτος master έναν όρο Stop που τοποθετεί το bus σε ένα Idle –State.



Εικόνα 3.37 : Μετάδοση σε ένα I^2C bus (Εγγραφή – Ανάγνωση)

3.3.1.3 – Όροι μετάδοσης του I^2C της Phillips



Εικόνα 3.38 : Μετάδοση σε ένα I^2C bus – ACK bit

- **Free (ΕΛΕΎΘΕΡΟ)** - το bus είναι ελεύθερο ή μη απασχολημένο - η γραμμή SDA δεδομένων και το ρολόι SCL είναι και τα δύο σε υψηλό επίπεδο..
- **Start Condition (ΈΝΑΡΞΗ)** ή Repeated Start (ΕΠΑΝΕΝΑΡΞΗ) - η μεταφορά δεδομένων αρχίζει με έναν όρο έναρξης. Το επίπεδο της γραμμής στοιχείων SDA αλλάζει από υψηλό σε χαμηλό, ενώ η γραμμή ρολογιών SCL παραμένει υψηλή. Όταν αυτό εμφανίζεται, το bus γίνεται "busy".
- **Change (ΑΛΛΑΓΉ)** - ενώ η γραμμή ρολογιών SCL είναι χαμηλή, το κομμάτι δεδομένων που μεταφέρεται μπορεί να εφαρμοστεί στη γραμμή δεδομένων SDA από μια συσκευή αποστολής σημάτων. Κατά τη διάρκεια αυτής της περιόδου, η γραμμή δεδομένων SDA μπορεί να αλλάξει την κατάστασή της, εφ' όσον η γραμμή SCL παραμένει χαμηλή.
- **Data (ΔΕΔΟΜΕΝΑ)** - ένα υψηλό ή χαμηλό κομμάτι των πληροφοριών για τη γραμμή στοιχείων SDA ισχύει κατά τη διάρκεια του υψηλού επιπέδου της γραμμής ρολογιών SCL. Αυτό το επίπεδο πρέπει να κρατηθεί σταθερό κατά τη διάρκεια

ολόκληρου του χρόνου που το ρολόι παραμένει υψηλό για να αποφευχθεί η παρερμηνεία ως όρος έναρξης ή λήξεως.

- **Stop Condition (ΛΗΞΗ)** - η μεταφορά στοιχείων ολοκληρώνεται από έναν όρο λήξης. Αυτό εμφανίζεται όταν περνά το επίπεδο στη γραμμή δεδομένων SDA από το χαμηλό κράτος στο υψηλό επίπεδο, ενώ η γραμμή ρολογιών SCL παραμένει υψηλή. Όταν η μεταφορά στοιχείων ολοκληρωθεί, το bus είναι πάλι ελεύθερο.

3.3.2 – Το STA013

Το STA013 είναι ένα πλήρως ολοκληρωμένο IC αποκωδικοποίησης mp3 που προσφέρει μεγάλη ευελιξία στην σχεδίαση. Το STA013 λαμβάνει τα δεδομένα με ένα σειριακό interface εισόδου και το αποκωδικοποιημένο σήμα στην έξοδο του μπορεί να είναι στερεοφωνικό, μονοφωνικό ή dual channel που μπορεί άμεσα να σταλθεί σ' ένα DAC μέσω του PCM interface εξόδου. Το interface εξόδου είναι προγραμματιζόμενο για να μπορεί να προσαρμοστεί στις περισσότερες συσκευές DAC που υπάρχουν στην αγορά.

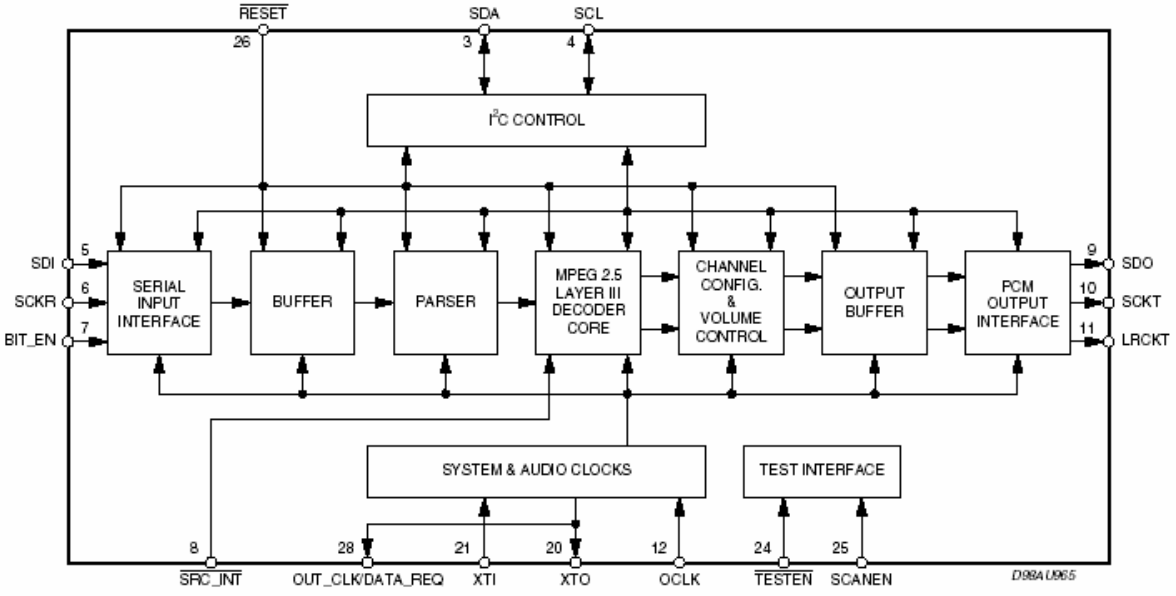
3.3.2.1 – Χαρακτηριστικά του STA013

Το STA013 έχει τα παρακάτω χαρακτηριστικά

- Υποστηρίζει αποκωδικοποίηση των μορφών συμπίεσης MPEG-1, MPEG-2, MPEG-2.5.
- Αποκωδικοποιεί σε στέρεο, μονοφωνικό και dual channel.
- Υποστηρίζει όλες τις συχνότητες MPEG-1, MPEG-2, MPEG-2.5.
- Μπορεί να δεχτεί δεδομένα με ρυθμό από 8 Kbps έως 320 Kbps.
- Πλήρης έλεγχος της έντασης της φωνής καθώς και ρύθμιση των μπάσων και των τρέμολων με ψηφιακό τρόπο.
- Αποκωδικοποίηση των δεδομένων μέσω I2C interface.
- PCM έξοδος που υποστηρίζει τις πιο γνωστές μορφές δεδομένων για άμεση αποστολή σε DAC.
- Μικρή κατανάλωση ισχύος.

3.3.2.2 – Αρχιτεκτονική του STA013

Η αρχιτεκτονική του STA013 φαίνεται παρακάτω:



Εικόνα 3.39 : Μπλοκ διάγραμμα του STA013

Όπως φαίνεται από το μπλοκ διάγραμμα το STA013 ελέγχεται μέσω I^2C interface (έχει 128 καταχωρητές I^2C). Ακόμα φαίνεται το σειριακό interface επικοινωνίας για την μεταφορά των δεδομένων. Ένας buffer για την αποθήκευση των δεδομένων επιτρέπει στο μικροελεγκτή που ελέγχει το STA013 να μην χρειάζεται να στέλνει συνεχώς δεδομένα στον αποκωδικοποιητή αλλά απλά να «γεμίζει» αυτό τον buffer όποτε απαιτείται και μέχρι αυτός να «αδειάσει» να μπορεί να ασχοληθεί με άλλες διεργασίες. Ακόμα φαίνεται η καρδιά του αποκωδικοποιητή που βασίζεται σε ένα DSP. Τέλος στο μπλοκ διάγραμμα φαίνεται και το PCM interface εξόδου που φροντίζει για την σωστή μορφή των ψηφιακών δεδομένων εξόδου.

3.3.2.3 – Pinout του STA013

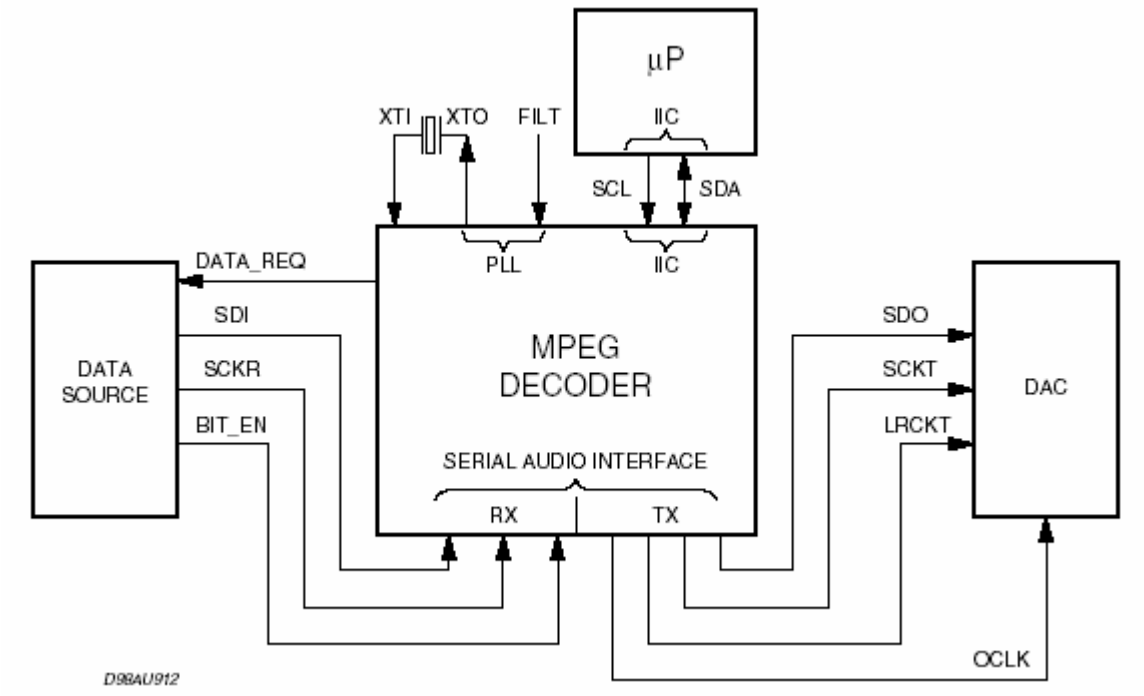
Οι είσοδοι και οι έξοδοι του STA013 φαίνονται στον παρακάτω πίνακα:

Πίνακας 3.13 : Pinout του STA013

SO28	TQFP44	LFBGA64	Pin Name	Type	Function	PAD Description
1	29	B5	VDD_1		Supply Voltage	
2	30	B4	VSS_1		Ground	
3	31	A4	SDA	I/O	i^2C Serial Data + Acknowledge	CMOS Input Pad Buffer CMOS 4mA Output Drive
4	32	B3	SCL	I	I^2C Serial Clock	CMOS Input Pad Buffer
5	34	A1	SDI	I	Receiver Serial Data	CMOS Input Pad Buffer
6	36	B2	SCKR	I	Receiver Serial Clock	CMOS Input Pad Buffer
7	38	D4	BIT_EN	I	Bit Enable	CMOS Input Pad Buffer with pull up
8	40	D1	$\overline{\text{SRC\_INT}}$	I	Interrupt Line For S.R. Control	CMOS Input Pad Buffer
9	42	E2	SDO	O	Transmitter Serial Data (PCM Data)	CMOS 4mA Output Drive
10	44	F2	SCKT	O	Transmitter Serial Clock	CMOS 4mA Output Drive
11	2	H1	LRCKT	O	Transmitter Left/Right Clock	CMOS 4mA Output Drive
12	3	H3	OCLK	I/O	Oversampling Clock for DAC	CMOS Input Pad Buffer CMOS 4mA Output Drive
13	5	F3	VSS_2		Ground	
14	6	E4	VDD_2		Supply Voltage	
15	7	G4	VSS_3		Ground	
16	8	G5	VDD_3		Supply Voltage	
17	10	F5	PVDD		PLL Power	
18	11	G6	PVSS		PLL Ground	
19	12	G7	FILT	O	PLL Filter Ext. Capacitor Conn.	
20	13	G8	XTO	O	Crystal Output	CMOS 4mA Output Drive
21	15	F7	XTI	I	Crystal Input (Clock Input)	Specific Level Input Pad (see paragraph 2.1)
22	19	E7	VSS_4		Ground	
23	21	C8	VDD_4		Supply Voltage	
24	22	D7	$\overline{\text{TESTEN}}$	I	Test Enable	CMOS Input Pad Buffer with pull up
25	24	A7	SCANEN	I	Scan Enable	CMOS Input Pad Buffer
26	25	B6	$\overline{\text{RESET}}$	I	System Reset	CMOS Input Pad Buffer with pull up
27	26	A5	VSS_5		Ground	
28	27	C5	OUT_CLK/ DATA_REQ	O	Buffered Output Clock/ Data Request Signal	CMOS 4mA Output Drive

3.3.2.4 – Interfaces του STA013

Το STA013 έχει τρία interfaces για την επικοινωνία του, ένα για τον έλεγχο του ένα για την τροφοδότησή του με δεδομένα και ένα για την έξοδο του. Αυτά φαίνονται στο παρακάτω σχηματικό διάγραμμα:



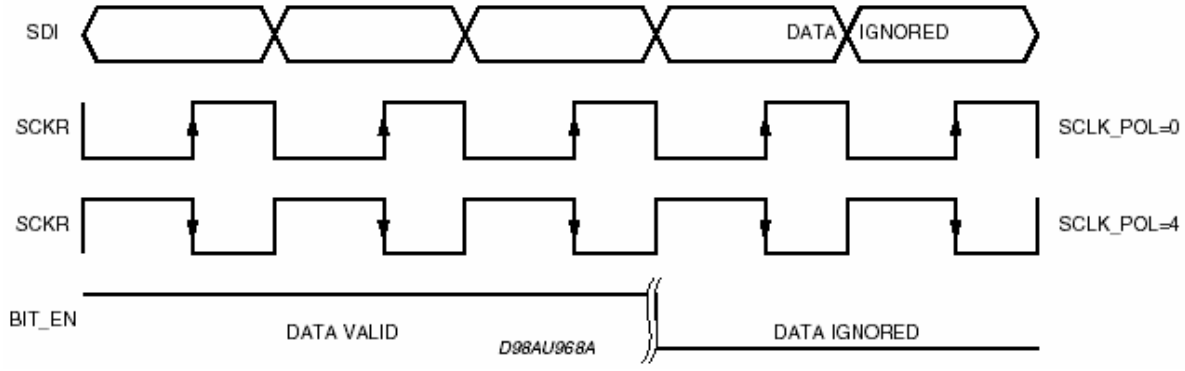
Εικόνα 3.40: Interfaces του STA013 - Τυπική συνδεσμολογία

3.3.2.4.1 – Το interface I^2C για τον έλεγχο του STA013

Το STA013 απαιτεί το I^2C για τον έλεγχο του. Έχει 128 I^2C καταχωρητές οι οποίοι και ελέγχονται μέσω του I^2C. Το interface έχει ήδη περιγραφεί εκτενώς.

3.3.2.4.2 – Το σειριακό interface για τα δεδομένα

Το σειριακό interface για τα δεδομένα που χρησιμοποιείται έχει την μορφή που δείχνει το παρακάτω σχήμα:

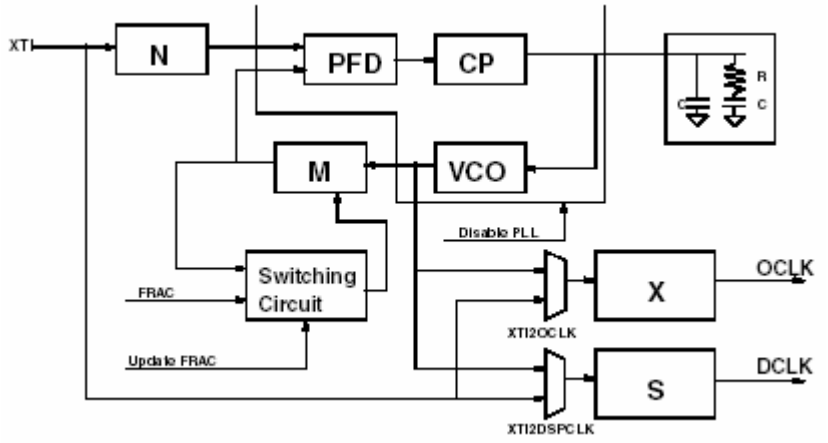


Εικόνα 3.41 : Σειριακό interface για τα δεδομένα

Όπως βλέπουμε κάθε bit δεδομένων μπορεί να σταλθεί τόσο στην άνοδο του ρολογιού τόσο και στην πτώση του ανάλογα με τις ρυθμίσεις που έχουν γίνει στο STA013.

Ακόμα το σήμα εισόδου BIT_ENABLE μπορεί να χρησιμοποιηθεί ώστε το STA013 να λαμβάνει ή να αγνοεί το εισερχόμενο ρεύμα δεδομένων.

Όταν το STA013 λαμβάνει ρολόι στην είσοδο του και ένα έγκυρο ρεύμα δεδομένων ενεργοποιείται το εσωτερικό PLL ρολόι, παρέχοντας στο DSP το ρολόι για την αποκωδικοποίηση των δεδομένων και στην έξοδο τις συχνότητες για τα δεδομένα.



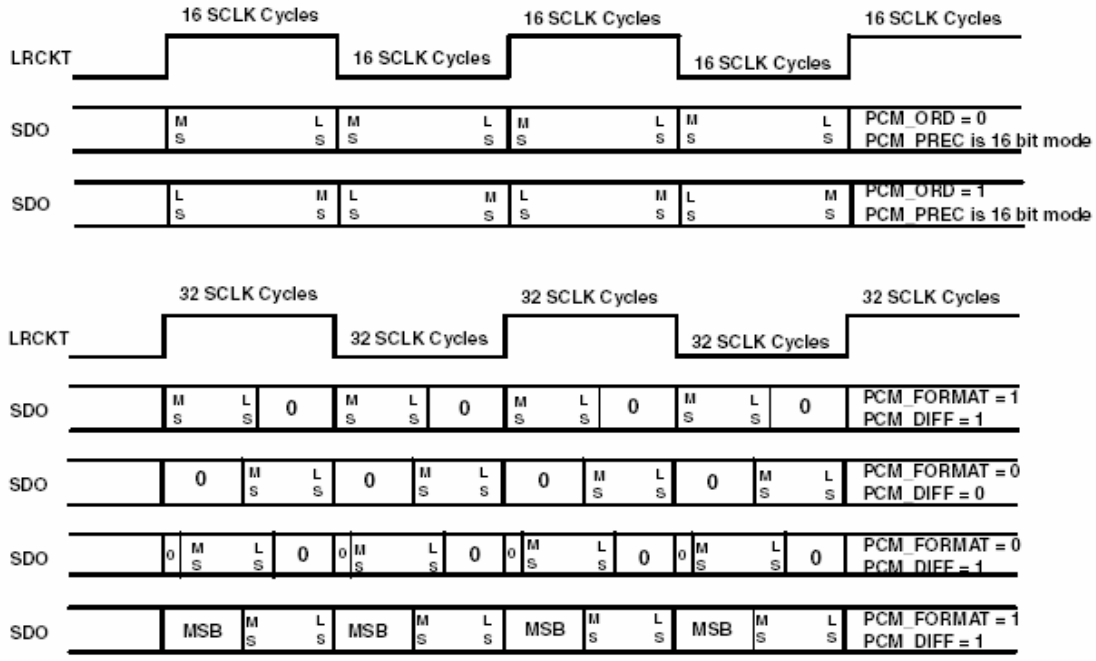
Εικόνα 3.42 : PLL και γεννήτρια ρολογιών

Οι συχνότητες δειγματοληψίας δημιουργούνται διαιρώντας το OCLK με προγραμματιζόμενους παράγοντες. Το PLL του STA013 μπορεί να οδηγήσει την πλειοψηφία των εμπορικών DACs.

3.3.2.4.3 – PCM interface εξόδου

Η ψηφιακή έξοδος του STA013 είναι σε σειριακή PCM μορφή. Αυτή αποτελείται από τα εξής σήματα:

1. SDO : PCM σειριακή έξοδος δεδομένων
2. SCKT : PCM σειριακή έξοδος ρολογιού
3. LRCLK : Ρολόι επιλογής καναλιού



Εικόνα 3.43 : Μορφές εξόδου PCM

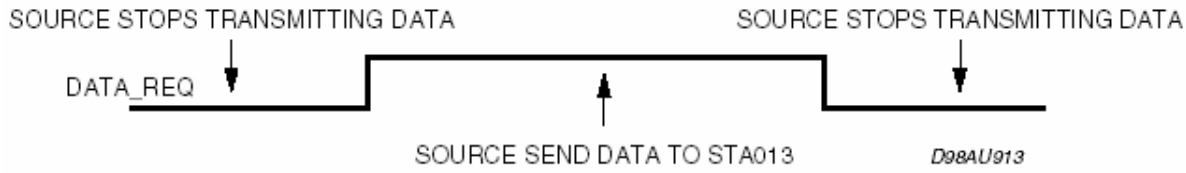
Η έξοδος μπορεί να έχει ακρίβεια από 16 έως 24 bits / word. Ακόμα μπορεί να μεταδοθεί με το MSB ή το LSB πρώτα.

Παραπάνω φαίνονται κάποιες μορφές εξόδου που μπορούν να επιτευχθούν με το STA013.

3.3.2.5 – Λειτουργία του STA013

Το STA013 μπορεί να λειτουργήσει σε δύο modes. Από τις δύο θα αναλυθεί η Multimedia mode η οποία και χρησιμοποιήθηκε. Σε αυτή την λειτουργία το STA013 αποκωδικοποιεί τα δεδομένα που εισέρχονται λειτουργώντας σαν master της διακίνησης δεδομένων από την πηγή στον εαυτό του. Ο έλεγχος γίνεται από μία ειδική διαχείριση από το ενσωματωμένο λογισμικό του STA013.

Το STA013 δηλώνει ότι χρειάζεται περισσότερα δεδομένα με το σήμα DATA_REQUEST. Όταν το STA013 ενεργοποιεί το σήμα DATA_REQUEST (αυτό μπορεί να σημαίνει είτε λογικό 1 είτε λογικό 0 καθώς η πολικότητα του είναι προγραμματιζόμενη) τότε η πηγή δεδομένων πρέπει να στείλει τα δεδομένα. Παρακάτω φαίνεται ένα παράδειγμα των προηγουμένων.



Εικόνα 3.44 : Μεταφορά δεδομένων – DATA_REQUEST σήμα

3.3.2.6 – Καταστάσεις λειτουργίας

Υπάρχουν τρεις διαφορετικές καταστάσεις λειτουργίας που μπορεί να βρίσκεται ο αποκωδικοποιητής ενώ είναι σε Multimedia mode. Αυτές είναι Idle, Init και Decode.

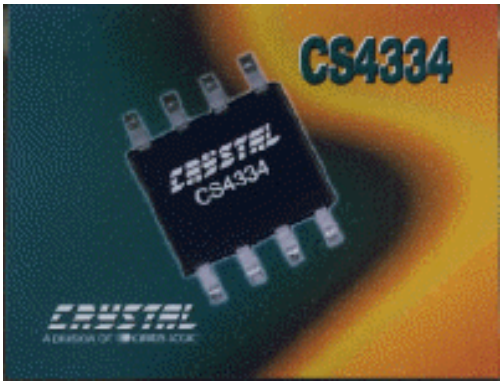
- **Idle:** Σε αυτή την κατάσταση ο αποκωδικοποιητής περιμένει την εντολή Run. Σε αυτή την κατάσταση μπορούν να γίνουν οι ρυθμίσεις του STA013.
- **Init:** Σε αυτή την κατάσταση ο αποκωδικοποιητής αγνοεί τις εντολές PLAY και MUTE. Η κατάσταση αυτή τελειώνει όταν τα πρώτα δεδομένα φτάσουν στην έξοδο.
- **Decode:** Αυτή η κατάσταση περιγράφεται πλήρως από τον παρακάτω πίνακα

Πίνακας 3.14 : Decoding κατάσταση του STA013

PLAY	MUTE	Clock State	PCM Output	Decoding
0	0	Not Running	0	No
0	1	Running	0	No
1	0	Running	Decoded Samples	Yes
1	1	Running	0	Yes

3.4 – Ο αναλογικό – σε – ψηφιακό μετατροπέας CS4334

Για την μετατροπή του ψηφιακού ήχου που εξάγεται από το STA013 σε αναλογικό χρειάζεται η χρησιμοποίηση ενός DAC. Για τον σκοπό αυτό επιλέχτηκε ο CS4334 της Cirrus Logic ο οποίος εκτός από τα πολύ καλά χαρακτηριστικά του στην μετατροπή του ήχου συνεργάζεται άψογα και με το STA013.



Εικόνα 3.45 : Ο DAC CS4334

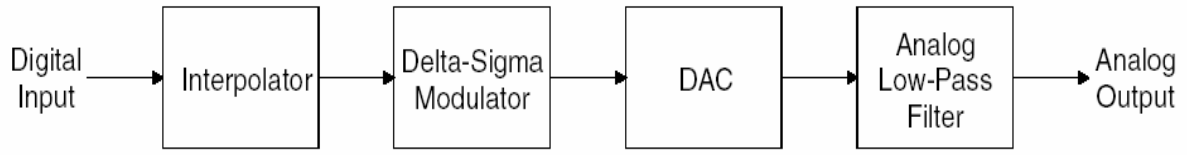
3.4.1 – Χαρακτηριστικά

Ο DAC CS4334 έχει τα παρακάτω τεχνικά χαρακτηριστικά:

- Πλήρης DAC λειτουργία
- 24-bit μετατροπή
- 96db δυναμικό εύρος
- -88db απόρριψη θορύβου
- Μονή τροφοδοσία +5V
- Φιλτραρισμένες γραμμές εξόδου
- On-Chip Digital De-emphasis

3.4.2 – Γενική περιγραφή

Η οικογένεια των CS4334 DACs προσφέρει πλήρη στέρεο μετατροπή από ψηφιακό σε αναλογικό, περιλαμβάνοντας ψηφιακή παρεμβολή, τέταρτης τάξης ΔΣ ψηφιακό σε αναλογικό μετατροπή, digital de-emphasis και αναλογικό φίλτρο στην έξοδο.



Εικόνα 3.46 : Μπλοκ διάγραμμα του DAC CS4334

Η οικογένεια των CS4334 DACs μπορεί να λειτουργήσει σε δύο modes. Την BRM στην οποία MCLK / LCLK είναι 256, 384 ή 512 και την HRM στην οποία αυτός ο λόγος είναι 128 ή 192. Στην HRM ο DAC μπορεί να υποστηρίξει μέχρι 100 KHz συχνότητες δειγματοληψίας.

3.4.2.1 – Ψηφιακό φίλτρο πόλωσης

Το ψηφιακό φίλτρο αυξάνει την συχνότητα δειγματοληψίας κατά 4 και ακολουθείται από ένα ψηφιακό φίλτρο 32x sample-and-hold (16x στην HRM). Αυτό γίνεται για την εύκολη απομάκρυνση των αρμονικών που προκαλούν θόρυβο και βρίσκονται σε συχνότητες πολλαπλάσιες της συχνότητας δειγματοληψίας.

3.4.2.2 – Δέλτα – Σίγμα Μετατροπέας

Ο ΔΣ μετατροπέας μετατρέπει την έξοδο του φίλτρου σε 1 bit δεδομένων με συχνότητα 128 x Fs. (64 x Fs στην HRM)

3.4.2.3 – DAC

Ο ΔΣ μετατροπέας ακολουθείται από ένα ψηφιακό σε αναλογικό μετατροπέα που μετατρέπει τα bits των δεδομένων σε πακέτα φόρτισης. Η τεχνική που χρησιμοποιείται μειώνει την ευαισθησια στον ρολόι και προσφέρει βαθυπερατό φιλτράρισμα στον ήχο.

3.4.2.4 – Αναλογικό βαθυπερατό φίλτρο

Το τελευταίο στάδιο είναι ένα αναλογικό βαθυπερατό φίλτρο που βοηθά στην απαλότητα του ήχου και στην αποκοπή του θορύβου σε συχνότητες εκτός της συχνότητας του ήχου.

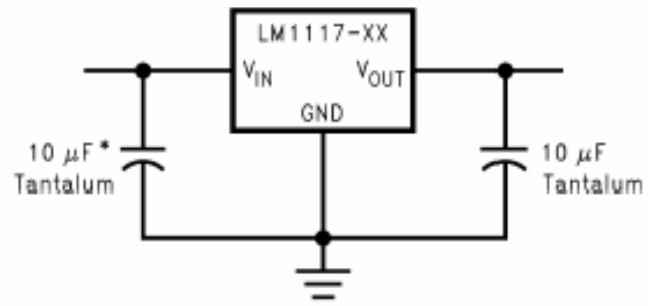
3.5 – Υπόλοιπα περιφερειακά του συστήματος

Για την ολοκλήρωση της κατασκευής χρησιμοποιήθηκαν ακόμα μερικά στοιχεία που αναφέρονται συνοπτικά παρακάτω.

3.5.1 – Voltage regulators

Η πλακέτα απαιτεί για την τροφοδοσία όλων των ολοκληρωμένων 3 και 5 Volts. Για αυτό το λόγο πρέπει να δημιουργηθούν αυτές οι τάσεις για να υπάρχει μόνο μία συνολική τροφοδοσία στο κύκλωμα. Ακόμα θεωρήθηκε καλό να δίνεται η δυνατότητα η τροφοδοσία να δίνεται από ένα μετασχηματιστή και όχι από γεννήτρια. Για αυτό το λόγο χρησιμοποιήθηκε ο LM-1117 voltage regulator της National.

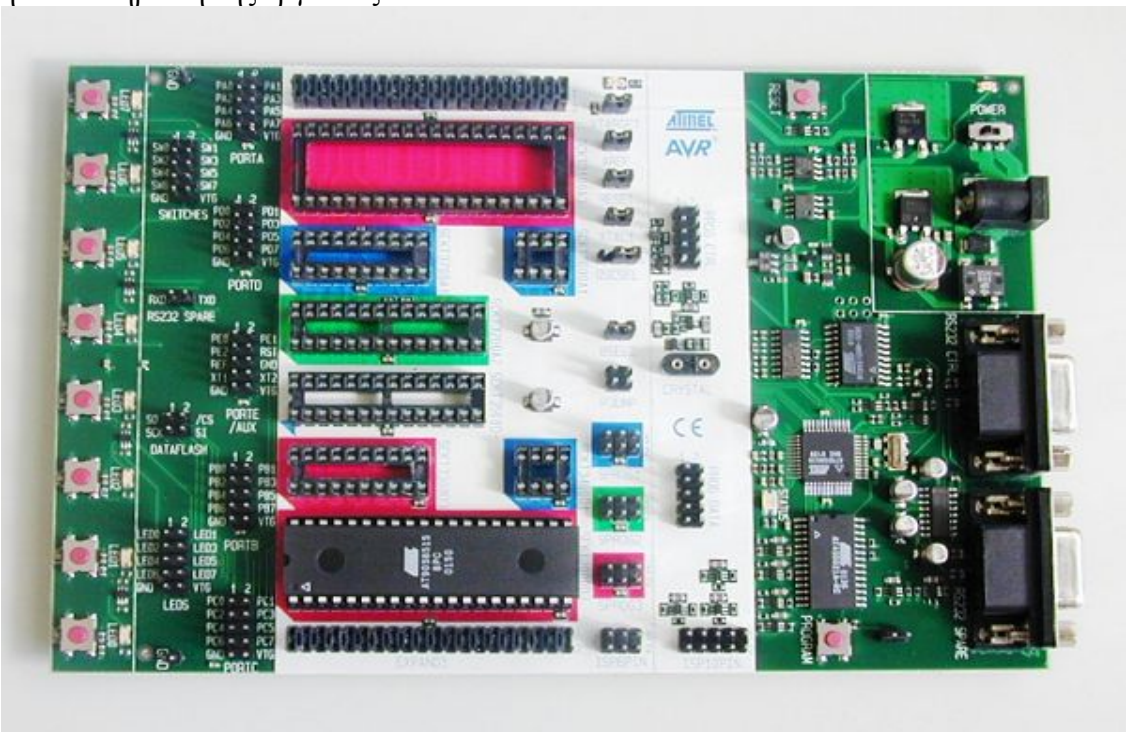
Ο LM-1117 χρησιμοποιείται στην fixed κατασκευή του για εξόδους 3V και 5V. Έτσι με μία απλή τροφοδοσία της τάξεως των 7,5 V τροφοδοτούνται όλα τα ολοκληρωμένα στην πλακέτα.



Εικόνα 3.47 : Τυπική συνδεσμολογία LM-1117

3.5.2 – Υπόλοιπα στοιχεία που χρησιμοποιήθηκαν

Τέλος για την εργασία χρησιμοποιήθηκαν πυκνωτές, αντιστάσεις, dip-switches, adapters και το αναπτυξιακό STK500 της ATMEL το οποίο αποτέλεσε μεγάλη βοήθεια στην ολοκλήρωση της εργασίας.



Εικόνα 3.48 : Το αναπτυξιακό STK500 της ATMEL

Κεφάλαιο 4

Το σύστημα αρχείων FAT16

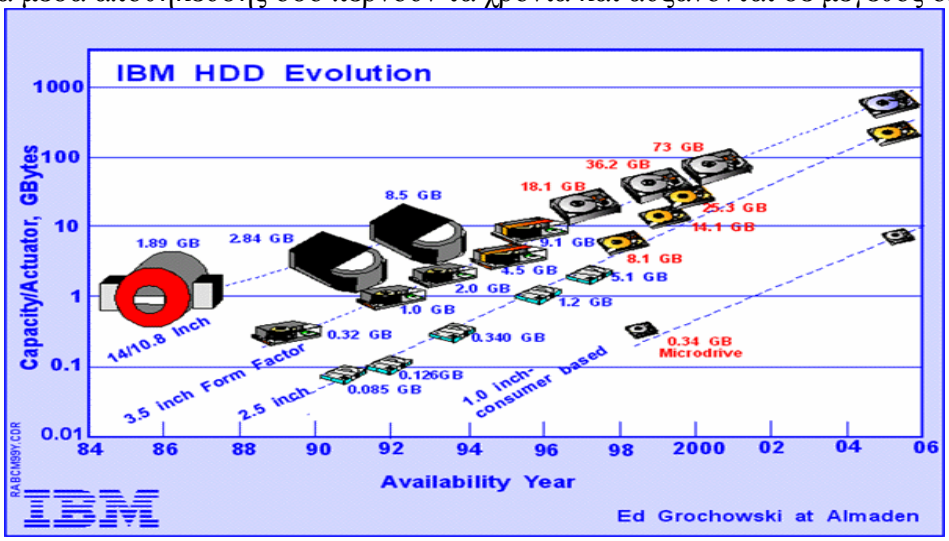
4.1 – Σύστημα Αρχείων -Εισαγωγη

Ο σκληρός δίσκος είναι, φυσικά, ένα μέσο για την αποθήκευση πληροφοριών. Οι σκληροί δίσκοι αυξάνονται στο μέγεθος κάθε έτος, και καθώς γίνονται μεγαλύτεροι, η χρησιμοποίηση τους με έναν αποδοτικό τρόπο γίνεται δυσκολότερη. Το σύστημα αρχείων είναι το γενικό όνομα που δίνεται στις λογικές δομές και στις ρουτίνες λογισμικού που χρησιμοποιούνται για να ελέγξουν την αποθήκευση στον σκληρό δίσκο. Τα λειτουργικά συστήματα χρησιμοποιούν διαφορετικούς τρόπους για την πρόσβαση στην αποθήκευση σε ένα σκληρών δίσκων, και αυτή η επιλογή είναι βασικά ανεξάρτητη από τη χρησιμοποίηση του συγκεκριμένου υλικού -- ο ίδιος σκληρός δίσκος μπορεί να τακτοποιηθεί με πολλούς διαφορετικούς τρόπους, και ακόμη και πολλαπλάσιους τρόπους στις διαφορετικές περιοχές του ίδιου δίσκου. Οι πληροφορίες για αυτό το τμήμα διαπερνούν στην πραγματικότητα τη λεπτή γραμμή μεταξύ του υλικού και του λογισμικού, μια γραμμή που γίνεται όλο και περισσότερο αδιαφανής κάθε έτος. Η φύση των λογικών δομών στον σκληρό δίσκο έχει μια σημαντική επιρροή στην απόδοση, την αξιοπιστία, επεκτασιμότητα και τη συμβατότητα του υποσυστήματος αποθήκευσής .

Για την εργασία τούτη αποφασίστηκε να χρησιμοποιηθεί ένα υπάρχον σύστημα αρχείων για την οργάνωση των αρχείων MP3 που βρίσκονται μέσα στην κάρτα τόσο γιατί δεν απαιτεί ιδιαίτερο επιπλέον προγραμματιστικό κόπο για την δημιουργία προγράμματος για την ανάγνωση του FAT μέσω του μικροελεγκτή ενώ είναι πλέον ιδιαίτερα εύκολο για το χρήστη να φορτώσει τα τραγούδια του μέσα στην μνήμη της συσκευής χωρίς να χρειάζεται πρόσθετες εφαρμογές. Σε αυτό το τμήμα, λοιπόν, ρίχνεται μια ματιά στις λογικές δομές στον σκληρό δίσκο στον τρόπο που ιδρύονται και χρησιμοποιούνται σε μια χαρακτηριστική εγκατάσταση στο PC. Η περιπλάνηση μας στα συστήματα αρχείων αρχίζει με μια σύντομη ιστορική αναδρομή στην εξέλιξη των χρησιμοποιούμενων συστημάτων αρχείων και συνεχίζει με την αναλυτική περιγραφή της δομής του FAT και ειδικώς του FAT16 που χρησιμοποιήθηκε.

4.2 – Σύντομη περιγραφή υπαρχόντων συστημάτων αρχείων

Τα μέσα αποθήκευσης όσο περνούν τα χρόνια και αυξάνονται σε μέγεθος όπως



Εικόνα 4.1 : Εξέλιξη χωρητικότητας σκληρών δίσκων

φαίνεται και στο παραπάνω σχήμα. Το μέγεθος ενός σκληρού δίσκου παίζει σημαντικότατο ρόλο στην δημιουργία του κατάλληλου συστήματος αρχείων (ειδικότερα στα παλαιά συστήματα με την λιγοστή μνήμη τα συστήματα αρχείων ήταν κατάλληλα ώστε να μην καταλαμβάνουν πολύ χώρο). Ειδικότερα, τα παλαιά συστήματα με τη λιγοστή μνήμη χρειάζονταν λιγότερα bits για την περιγραφή όλων των αρχείων στο σύστημα αλλά είχαν παράλληλα την απαίτηση το σύστημα αρχείων να μην καταλαμβάνει πολύ χώρο. Αυτή η απαίτηση έχει αφήσει ακόμα και στις μέρες μας κάποια κατάλοιπα όπως περίεργες κωδικοποιήσεις σε κάποια στοιχεία του συστήματος αρχείων. Παρακάτω γίνεται σύντομη αναφορά στα κυριότερα συστήματα αρχείων ενώ ακολουθεί αναλυτική περιγραφή του FAT16 που χρησιμοποιήθηκε στην εργασία.

4.2.1 – Οικογένεια συστημάτων αρχείων FAT

Το πιο κοινό σύστημα αρχείων στον κόσμο των υπολογιστών είναι στην πραγματικότητα μια οικογένεια συστημάτων αρχείων. Το βασικό όνομα για αυτό το σύστημα αρχείων είναι FAT, το όνομα προέρχεται από μια από τις κύριες λογικές δομές που το σύστημα αρχείων χρησιμοποιεί: ο πίνακας κατανομής αρχείων (File Allocation Table). Αυτό το σύστημα αρχείων είναι αυτό που χρησιμοποιήθηκε από το DOS στα πρώτα IBM PCs, και έγινε το πρότυπα για τα PCs που ακολούθησε. Σήμερα, τα περισσότερα PCs χρησιμοποιούν ακόμα μια παραλλαγή του βασικού FAT συστήματος αρχείων που δημιουργήθηκε για εκείνες τις πρόωρες μηχανές πάνω από δύο δεκαετίες πριν. Το κοινό όνομα του συστήματος αρχείων, "FAT", είναι προβληματικό, παρόλο που ακόμα συχνά χρησιμοποιείται.

Το πρώτο FAT σύστημα αρχείων χρησιμοποίησε τους πίνακες κατανομής αρχείων 12-bit αυτό επεκτάθηκε αργότερα σε 16 bit, και έγινε η πιο κοινή εφαρμογή συστημάτων αρχείων για τους σκληρούς δίσκους από την πρόσφατη δεκαετία του '80 έως την πρόσφατη δεκαετία του '90 Για να διακριθούν αυτές τις εκδόσεις του FAT από τον τριανταδυάμπιτο διάδοχο αποκαλούμενο FAT32, οι παλαιότερες παραλλαγές του FAT τώρα μερικές φορές καλούνται FAT12 ή FAT16. Εντούτοις, θα ακούσετε ακόμα και μόνο "FAT" χρησιμοποιούμενο πολύ, σε αυτή την περίπτωση πρέπει να ανακαλύψετε σε τι συγκεκριμένα αναφέρεται, εάν χρειάζεται σε εκείνο το ιδιαίτερο πλαίσιο. Περισσότερη επεξεργασία στις διαφορές μεταξύ FAT12, FAT16 και FAT32, αναφέρονται παρακάτω.

Ενώ σχεδιάστηκε συγκεκριμένα για το MS-DOS η χρησιμοποίηση του από την πλειοψηφία των λειτουργικών συστημάτων της Microsoft, έχει αναγκάσει την υποστήριξη του από τα λειτουργικά συστήματα που ακολούθησαν (με κάποιο τρόπο) με αποτέλεσμα να γίνει ένα σημαντικό χαρακτηριστικό για άλλα λειτουργικά συστήματα επίσης. Ουσιαστικά όλα τα λειτουργικά συστήματα PC μπορούν να υποστηρίξουν το βασικό FAT σύστημα αρχείων, ακόμη και αυτά που δείχνουν λίγη ομοιότητα στο DOS της Microsoft ή τα Windows. Οι FAT12 και FAT16 partitions ενώ είναι ο "μικρότερος κοινός παρονομαστής" των συστημάτων αρχείων -- αφού περιορίζονται από πολλές απόψεις, είναι τα ευκολότερα συστήματα αρχείων για να χρησιμοποιηθούν αφού η συμβατότητα δεν είναι πρόβλημα. Ακόμη και οι πλατφόρμες μη-PC μπορούν σε πολλές περιπτώσεις να διαβάσουν τους δίσκους που σχηματοποιούνται στο FAT σύστημα αρχείων.

4.2.2 – VFAT – Μια παραλλαγή του FAT

Όταν η Microsoft εισήγαγε τα WINDOWS.95, έκανε διάφορες βελτιώσεις στο λειτουργικό σύστημα σε σύγκριση με τους προκατόχους του. Μια από τις αλλαγές που έγιναν ήταν μια αύξηση στο κλασσικό σύστημα αρχείων FAT16 που ήταν σε χρήση μέχρι εκείνο το σημείο. Η νέα παραλλαγή του FAT κλήθηκε το εικονικό FAT ή VFAT για συντόμευση. (Σημειώστε ότι πολλοί άνθρωποι χρησιμοποιούν τους όρους "FAT" και "VFAT" εναλλακτικά, ακόμα κι αν δεν είναι τεχνικά οι ίδιοι. Όλες οι "FAT" partitions που

δημιουργούνται στο πλαίσιο των WINDOWS.95 ή των πιο πρόσφατων λειτουργικών συστημάτων είναι στην πραγματικότητα partitions VFAT.)

Το VFAT έχει διάφορα χαρακτηριστικά και βελτιώσεις έναντι σε FAT12 και FAT16:

- Μακροχρόνια υποστήριξη ονόματος αρχείων: Ένας από τους πιο ενοχλητικούς περιορισμούς των λειτουργικών συστημάτων πριν από τα WINDOWS.95 ήταν ο περιορισμός ονόματος αρχείων στους ένδεκα χαρακτήρες. Για τους περισσότερους ανθρώπους, η σημαντικότερη βελτίωση του VFAT ήταν ότι επέτρεπε τη χρήση των μεγάλων ονομάτων αρχείων από το σύστημα και τις εφαρμογές των WINDOWS.95 , διατηρώντας παράλληλα τη συμβατότητα με το παλαιότερο λογισμικό που είχε γραφτεί προτού να εφαρμοστεί το VFAT.
- Βελτιωμένη απόδοση: Οι ρουτίνες πρόσβασης στο δίσκο και διαχείρισης αρχείων για VFAT ξαναγράφηκαν χρησιμοποιώντας τον τριανταδυάμπιτο κώδικα για να βελτιώσουν την απόδοση. Συγχρόνως, ο δεκαεξάμπιτος κώδικας διατηρήθηκε, για τη χρήση σε περίπτωση ανάγκης αλλά και για τη συμβατότητα.
- Καλύτερες ικανότητες διαχείρισης του συστήματος: Η ειδική υποστήριξη που προστέθηκε καθώς και οι τεχνικές όπως το κλείδωμα δίσκων, ήταν μεγάλη βοήθεια για τον χρήστη.

Παρά το νέο όνομα και τις νέες ικανότητες, το VFAT ως σύστημα αρχείων είναι βασικά το ίδιο με το FAT. Οι περισσότερες από τις νέες ικανότητες αφορούν το πώς το σύστημα αρχείων χρησιμοποιείται, και όχι οι πραγματικές δομές στο δίσκο που η μόνη σημαντική αλλαγή από την άποψη των πραγματικών δομών είναι η προσθήκη των μεγάλων ονομάτων αρχείων.

4.2.3 – 32-bit FAT – FAT32

Δεδομένου ότι οι σκληροί δίσκοι συνέχισαν να αυξάνονται στο μέγεθος στη δεκαετία του '90, οι περιορισμοί των συστημάτων αρχείων FAT16 και VFAT άρχισαν να γίνονται προφανείς. Η χρήση των μεγάλων μεγεθών cluster, οδήγησε σε ένα σημαντικό ποσό σπαταλημένου χώρου των σκληρών δίσκων (νωθρού). Τελικά, οι κατασκευαστές σκληρών δίσκων άρχισαν να δημιουργούν δίσκους τόσο μεγάλους που το FAT16 δεν θα μπορούσε να χρησιμοποιηθεί για να φορμάρει ολόκληρο το δίσκο σε ένα ενιαίο partition. Οι κατασκευαστές PC παραπονέθηκαν ότι FAT16 ήταν αδέξιο για τις σύγχρονες μηχανές, ενώ και οι χρήστες μπερδεύτηκαν με PCs που είχαν δύο ή τρεις διαφορετικούς "σκληρούς δίσκους" (που ήταν φυσικά στην πραγματικότητα διαφορετικά partitions που δημιουργήθηκαν για να αντεπεξέλθουν με το μέγιστο μέγεθος δίσκου που επέτρεπε το FAT16.)

Για να διορθώσει αυτήν την κατάσταση, η Microsoft δημιούργησε το FAT32. Αυτή η νεώτερη FAT παραλλαγή είναι μια αύξηση του VFAT συστήματος αρχείων (ακόμα κι αν το "V" παραλείφθηκε από το καινούργιο όνομα, το FAT32 είναι βασισμένο περισσότερο στο VFAT παρά στο FAT). Ονομάζεται FAT32 επειδή χρησιμοποιεί τριανταδυάμπιτους αριθμούς για να αντιπροσωπεύσει τα clusters, αντί των δεκαεξάμπιτων αριθμών που χρησιμοποιούνται από το FAT16. Το FAT32 δημιουργήθηκε πρώτιστα για να λύσει τα δύο προαναφερθέντα προβλήματα. Επιτρέπει την δημιουργία ενός και μόνο partition πολύ μεγάλου μεγέθους, ενώ το FAT16 περιορίζεται σε partitions μέγιστου μεγέθους περίπου 2 GB. Ελαχιστοποιεί επίσης το σπαταλημένο χώρο όταν συγκρίνεται με τα partitions του FAT16, επειδή χρησιμοποιεί πολύ μικρότερα μεγέθη cluster από το FAT16.

Το FAT32 εισήχθη αρχικά στην δεύτερη έκδοση των WINDOWS.95, και ήταν διαθέσιμο μόνο στις πιο πρόσφατες εκδόσεις των WINDOWS.95 όταν αγοράζονταν από

έναν κατασκευαστή υλικού. Η υποστήριξη FAT32 περιλήφθηκε αργότερα στα Windows 98, τα Windows Me και τα Windows2000 επίσης. Πολλά λειτουργικά συστήματα μη-Microsoft μπορούν επίσης τώρα είτε να διαβάσουν από το FAT32, είτε έχουν δυνατότητα ανάγνωση-εγγραφής partitions σε FAT32. Με τα μεγέθη σκληρών δίσκων να διευθύνονται στη στρατόσφαιρα, το FAT32 έχει σχεδόν αντικαταστήσει το FAT16 για εκείνους που χρησιμοποιούν τα λειτουργικά συστήματα της Microsoft "καταναλωτικού βαθμού".

4.2.4 – Νέας τεχνολογίας σύστημα αρχείων (NTFS version 1.1 – 4.0)

Όταν η Microsoft δημιούργησε τα WINDOWS NT, έχτισε το λειτουργικό σύστημα λίγο πολύ από την αρχή -- βασίστηκε σε ορισμένες υπάρχουσες έννοιες, φυσικά, αλλά ήταν συνολικά διαφορετικό από τα παλαιότερα λειτουργικά συστήματα της Microsoft. Ένα από τα βασικά στοιχεία της αρχιτεκτονικής NT ήταν το σύστημα αρχείων που δημιουργήθηκε ειδικά για το λειτουργικό σύστημα, αποκαλούμενο το σύστημα αρχείων νέας τεχνολογίας ή NTFS. Το NTFS επιτρέπει σε πολλούς από τους στόχους των WINDOWS NT να πραγματοποιηθούν.

Το NTFS είναι ένα πιο σύνθετο και ικανό σύστημα αρχείων από οποιοδήποτε από την οικογένεια των συστημάτων αρχείων FAT. Σχεδιάστηκε για το εταιρικό και επιχειρησιακό περιβάλλον, χτίστηκε για τη δικτυακή χρήση με κύριους στόχους την ασφάλεια, την αξιοπιστία και την αποδοτικότητα. Περιλαμβάνει πολλά καινούργια χαρακτηριστικά γνωρίσματα, συμπεριλαμβανομένου της συμπίεσης αρχείων, του πλήρους ελέγχου αδειών και ιδιοτήτων, την υποστήριξη για τα πολύ μεγάλα αρχεία, και στην συναλλαγή-βασισμένη στην λειτουργία. Επίσης δεν έχει τα προβλήματα μεγέθους με τους περιορισμούς μεγέθους των clusters και των σκληρών δίσκων όπως το FAT, και έχει άλλα χαρακτηριστικά γνωρίσματα που ενισχύουν την απόδοση όπως η υποστήριξη του RAID. Τα σημαντικότερα μειονεκτήματά του είναι η αυξανόμενη πολυπλοκότητα, και η λιγότερη συμβατότητα με άλλα λειτουργικά συστήματα έναντι στο FAT.

Το σύστημα αρχείων NTFS έχει στην πραγματικότητα περισσότερες από μια εκδόσεις. Μία που χρησιμοποιείται από τα WINDOWS NT καλείται συνήθως είτε έκδοση 1.1 είτε έκδοση 4.0, και έχει λιγότερα χαρακτηριστικά γνωρίσματα από το νεώτερο NTFS 5,0 που χρησιμοποιείται από τα WINDOWS 2000.

Ενώ δημιουργείται αρχικά για τα WINDOWS NT και τώρα χρησιμοποιείται πρώτιστα από τα WINDOWS NT και το διάδοχό τους, WINDOWS 2000, περιορισμένη υποστήριξη NTFS έχει προστεθεί επίσης σε άλλα λειτουργικά συστήματα. Παραδείγματος χάριν, λειτουργικά συστήματα όπως το Linux (καθώς επίσης και μερικές άλλες παραλλαγές Unix) και BeOS μπορούν να διαβάσουν τα partitions NTFS, το οποίο βοηθά με τη λειτουργικότητα των διαφορετικών λειτουργικών συστημάτων στην ίδια μηχανή. Εντούτοις, τα λειτουργικά συστήματα μη-Microsoft δεν μπορούν συνήθως να γράψουν στα partitions NTFS, γενικά λόγω των χαρακτηριστικών ασφάλειας που εισάγονται στο NTFS από τη Microsoft.

4.2.5 – Νέας τεχνολογίας σύστημα αρχείων (NTFS version 5.0)

Με τη δημιουργία των WINDOWS2000, η Microsoft πρόσθεσε διάφορα νέα χαρακτηριστικά γνωρίσματα στο λειτουργικό σύστημα, που έγινε γνωστό αρχικά ως WINDOWS NT 5.0. Αρκετά από αυτά τα χαρακτηριστικά γνωρίσματα είναι δεμένα πολύ με τα χαρακτηριστικά του συστήματος αρχείων, το οποίο η Microsoft ενημέρωσε συγχρόνως. Η νέα παραλλαγή NTFS καλείται NTFS 5,0. Στην πραγματικότητα, τα WINDOWS2000 στηρίζονται στις αλλαγές που το NTFS 5,0 περιλαμβάνει τόσο πολύ που το NTFS 5,0 απαιτείται. Έτσι το λειτουργικό σύστημα θα μετατρέψει τους παλαιότερους όγκους NTFS σε NTFS 5,0, και σε μερικές περιπτώσεις το NTFS πρέπει να υπάρχει για βασικά ικανότητες των WINDOWS 2000.

Το NTFS 5,0 περιλαμβάνει διάφορα νέα και χρήσιμα χαρακτηριστικά γνωρίσματα έναντι στις παλαιότερες εκδόσεις NTFS χρησιμοποιούμενες στα WINDOWS NT 4.0 και νωρίτερα. Το NTFS 5,0 σχεδιάστηκε για τα WINDOWS2000 και αυτό είναι το μόνο λειτουργικό σύστημα που παρέχει την πλήρη υποστήριξη για το σύστημα αρχείων. Τα WINDOWS NT 4.0 μπορούν επίσης να έχουν πρόσβαση στα partitions NTFS 5,0 εάν το λειτουργικό σύστημα ενημερώνεται με το service pack 4 (SP4), εντούτοις, δεν μπορεί να χρησιμοποιήσει τα νέα χαρακτηριστικά γνωρίσματα που NTFS 5,0.

4.2.6 - High Performance File System (HPFS)

Όταν Microsoft και Intel δημιουργούν το OS/2, επιδίωξαν να δημιουργήσουν ένα λειτουργικό σύστημα που είμαι περισσότερο ικανό από ότι το DOS και τα WINDOWS που ήταν τότε διαθέσιμα. Στην περίοδο αυτή, το μόνο σύστημα αρχείων που χρησιμοποιούνταν ευρέως στο PC ήταν το FAT σύστημα αρχείων, το οποίο είχε διάφορους σημαντικούς περιορισμούς. Το FAT σύστημα αρχείων περιοριζόταν από την άποψη του μεγέθους των partitions που θα μπορούσε να υποστηρίξει, επιτρέπονταν μόνο τα ονόματα 11-χαρακτήρων, και δεν είχε κανένα από τα χαρακτηριστικά γνωρίσματα οργάνωσης, ασφάλειας και αξιοπιστίας που είναι τόσο σημαντικά σε εταιρίες, επιχειρήσεις και τους μεμονωμένους χρήστες. Για να ικανοποιηθούν αυτές οι ανησυχίες, ένα νέο σύστημα αρχείων δημιουργήθηκε συγκεκριμένα για OS/2: το σύστημα αρχείων υψηλής απόδοσης ή αλλιώς HPFS.

Το HPFS προσφέρει πολλές σημαντικές βελτιώσεις σε σχέση με το FAT σύστημα αρχείων. Αυτές περιλαμβάνουν τα εξής:

- Υποστήριξη για τα μεγάλα ονόματα αρχείων (μέχρι 254 χαρακτήρες)
- Αποδοτικότερη χρήση του χώρου των δίσκων, δεδομένου ότι τα αρχεία αποθηκεύονται σε μια βάση ανά-τομέα αντί της χρησιμοποίησης των clusters πολλαπλάσιων του τομέα.
- Καλύτερη απόδοση λόγω της συνολικής σχεδίασης, συμπεριλαμβανομένης μιας εσωτερικής αρχιτεκτονικής με σκοπό να κρατήσει τα σχετικά στοιχεία πιό στενά το ένα στο άλλο στον χώρο των δίσκων.
- Λιγότερος τεμαχισμός των στοιχείων κατά τη διάρκεια του χρόνου, και λιγότερη ανάγκη στο defragment το σύστημα αρχείων.

Εξετάζοντας πόσο περισσότερο προηγμένο ήταν από το FAT, κάποιος πρέπει να αναμένει ότι το HPFS θα είχε γίνει αρκετά δημοφιλές. Δυστυχώς, το βαγόνι που ήταν προσκολλημένο το HPFS ήταν το OS/2, και για διάφορους λόγους (πολλοί από τους σχετικούς με την πολιτική μεταξύ της IBM και της Microsoft) το OS/2 ποτέ πραγματικά δεν έγινε δημοφιλές. Δεδομένου ότι το ενδιαφέρον για το OS/2 εξασθένισε, έτσι ελαχιστοποιήθηκε και η υποστήριξη για το HPFS. Ενώ οι πρώρες εκδόσεις των WINDOWS NT (3,51 και προηγούμενος) υποστήριζαν το HPFS, αυτή η υποστήριξη αφαιρέθηκε από την έκδοση των WINDOWS NT 4.0. και έπειτα. Υποστήριξη για όγκους HPFS μπορεί να προστεθεί σε μερικά μη- OS/2 λειτουργικά συστήματα με τη χρησιμοποίηση έξτρα λογισμικού, αλλά γενικώς αυτό το σύστημα αρχείων φαίνεται να είναι σταθερά σε μια σχέση φθοράς – αφθαρσίας. Πολλά από τα χαρακτηριστικά γνωρίσματά του ενσωματώθηκαν στο NTFS από τη Microsoft, κάποιος μπορεί να σημειώσει -- ή τουλάχιστον, το NTFS έχει μερικές ομοιότητες με το HPFS.

4.2.7 - UNIX / Linux συστήματα αρχείων

Όλα τα συστήματα αρχείων εκτελούν τις ίδιες βασικές λειτουργίες, βασισμένες στους θεμελιώδεις στόχους τους: η ευφυής οργάνωση των στοιχείων και ο αποδοτικός έλεγχος και πρόσβαση σε εκείνο το στοιχείο. Κατά συνέπεια, τα περισσότεροι από αυτά

μοιάζουν το ένα με το άλλο ως ένα ορισμένο βαθμό, ακόμα κι αν είναι επίσης διαφορετικά το ένα από το άλλο σε σημαντικούς τομείς. Ακόμα κι αν δύο συστήματα αρχείων PC διαφέρουν πολύ από την άποψη της εσωτερικών αρχιτεκτονικής και των δομών τους, μπορούν να γίνουν για να μοιάσουν με το ένα το άλλο πολύ στην εξωτερική εμφάνισή τους. Παραδείγματος χάριν, τα WINDOWS NT θα υποστηρίξουν και το FAT και τα partitions NTFS, που είναι συνολικά διαφορετικά από την άποψη των εσωτερικών δομών τους, αλλά έχουν ουσιαστικά το ίδιο interface για το χρήστη (και για τις εφαρμογές λογισμικού).

Το Unix ήταν στην αγορά για πολλές δεκαετίες, κάτι που το καθιστούν το παλαιότερο όλων των συστημάτων αρχείων που χρησιμοποιούνται στο PC. Τα συστήματα αρχείων Unix είναι επίσης πιθανώς τα πιό διαφορετικά από τα άλλα συστήματα αρχείων που χρησιμοποιούνται στο PC, και εσωτερικά και εξωτερικά (αναφερόμενος στο πώς ο χρήστης έχει πρόσβαση στο σύστημα αρχείων). Ενώ οι περισσότεροι χρήστες WINDOWS είναι εξοικειωμένοι με το interface εξερευνητής-τύπων για τη διαχείριση των αρχείων και των φακέλων, τα αρχεία Unix ρυθμίζονται συνήθως με τις ιδιαίτερες εντολές κειμένων -- παρόμοιες με το πώς το DOS λειτουργεί (στην πραγματικότητα, πολλές αρχές του FAT συστήματος αρχείων είναι βασισμένες στο Unix). Υπάρχουν γραφικά περιβλήματα (shells) Unix επίσης, φυσικά, αλλά πολλοί χρήστες Unix δεν τα χρησιμοποιούν ποτέ.

Οι σημαντικότερες διαφορές, εντούτοις, είναι εσωτερικές. Τα συστήματα αρχείων Unix σχεδιάζονται όχι για την εύκολη χρήση, αλλά για την ευρωστία, την ασφάλεια και την ευελιξία. Τα συστήματα αρχείων Unix προσφέρουν τα ακόλουθα χαρακτηριστικά γνωρίσματα, και έχουν ήδη για πολλά έτη:

- Άριστη επεκτασιμότητα, και υποστήριξη για τις μεγάλες συσκευές αποθήκευσης.
- Επιπέδου κατάλογου και αρχείου έλεγχοι ασφάλειας και πρόσβασης, συμπεριλαμβανομένης της δυνατότητας να ελεγχθεί ποιες χρήστες ή ομάδες χρηστών μπορούν να διαβάσουν, να γράψουν ή να εκτελέσουν ένα αρχείο.
- Πολύ καλή εκτέλεση και αποδοτική λειτουργία.
- Η δυνατότητα να δημιουργηθούν τα "εύκαμπτα" συστήματα αρχείων που περιέχουν πολλές διαφορετικές συσκευές, να συνδυαστούν οι συσκευές και να παρουσιαστούν ως ενιαίο σύστημα αρχείων, ή να τοποθετηθούν προσωρινά άλλες συσκευές αποθήκευσης για χρήση.
- Εγκαταστάσεις για αποτελεσματική χρησιμοποίηση από πολλούς χρήστες και προγράμματα σε ένα πολλαπλών καθηκόντων περιβάλλον, απαιτώντας ένα ελάχιστο έλεγχο.
- Τρόποι να δημιουργηθούν ειδικές δομές όπως τα λογικά συνδεμένα αρχεία.
- Χαρακτηριστικά γνωρίσματα αξιοπιστίας και ευρωστίας όπως η περιστροφή και η υποστήριξη για το RAID.

Εάν αυτά τα χαρακτηριστικά γνωρίσματα ηχούν παρόμοια με εκείνα του NTFS, είναι επειδή το Unix και τα Windows NT/2000 ανταγωνίζονται τώρα για ένα μεγάλο μέρος της ίδιας αγοράς, έτσι στο NTFS δόθηκε μεγάλο μέρος των ικανοτήτων που το Unix έχει. Υπάρχουν πολλά άλλα χαρακτηριστικά γνωρίσματα επίσης, τα οποία διαφέρουν από μια εφαρμογή σε άλλη -- δεν υπάρχει ένα "σύστημα αρχείων Unix", όπως και δεν υπάρχει ένα ενιαίο "λειτουργικό σύστημα Unix". Κάθε Unix παραλλαγή (συμπεριλαμβανομένης της δημοφιλούς Linux για το PC) έχει ένα ελαφρώς διαφορετικό σύστημα αρχείων, αν και αυτά είναι φυσικά πολύ παρόμοια μεταξύ τους, και είναι συνήθως δυνατός για διαφορετικά Unix συστήματα αρχείων να διαβάζουν ο ένας αρχεία του άλλου.

Τα συστήματα αρχείων Unix ήταν ένα από τα πρώτα (εάν όχι το πρώτο) που χρησιμοποίησαν την ιεραρχική δομή καταλόγου, με έναν κατάλογο ρίζας και φωλιασμένους υποκαταλόγους. (Οι περισσότεροι από μας εξοικειώνονται με αυτό από τη χρησιμοποίηση

του με το FAT σύστημα αρχείων, το οποίο λειτουργεί τον ίδιο τρόπο). Ένα από τα βασικά χαρακτηριστικά των συστημάτων αρχείων Unix είναι ότι ουσιαστικά όλα ορίζονται ως αρχείο -- τα κανονικά αρχεία κειμένων είναι φυσικά αρχεία, αλλά έτσι είναι και εκτελέσιμα προγράμματα, κατάλογοι, και ακόμη και οι συσκευές υλικού χαρτογραφούνται στα ονόματα αρχείων. Αυτό παρέχει την τεράστια ευελιξία στους προγραμματιστές και τους χρήστες, ακόμα κι αν υπάρχει μία δυσκολία εκμάθησης του πρώτα.

Δεδομένου ότι λίγοι χρήστες PC τρέχουν το Unix στις μηχανές τους, είναι απίθανο ότι πραγματικά θα χρησιμοποιήσετε ένα σύστημα αρχείων Unix άμεσα. Εντούτοις, μπορείτε να βρεθείτε σε ένα σύστημα αρχείων Unix εάν διαβάζετε μία ιστοσελίδα, παραδείγματος χάριν, καταλαβαίνοντας έτσι τα βασικά για το πώς η εργασία σε Unix είναι μια καλή ιδέα.

4.2.8 - BeOS File System (BFS)

Το λειτουργικό σύστημα BeOS σχεδιάστηκε για να χρησιμοποιήσει το σύστημα αρχείων του, αποκαλούμενο (δεν μας προκαλεί έκπληξη αυτό) σύστημα αρχείων BeOS, σε συντομία BFS ή befs μερικές φορές. Το BFS είναι βασισμένο έντονα στις θεμελιώδεις έννοιες σχεδιασμού των συστημάτων αρχείων Unix, έτσι εκείνοι που βιώνονται με το Unix θα βρουν τους εαυτούς τους πολύ εξοικειωμένους με το BFS. Εντούτοις, το BFS έχει προσαρμοστεί επίσης για να συναντήσει μερικούς από τους στόχους του λειτουργικού συστήματος, έτσι το BFS έχει επίσης μερικά ειδικά χαρακτηριστικά γνωρίσματα που απεικονίζουν τον αντικειμενικό σκοπό της Be της κατασκευής του BeOS ως φιλικό λειτουργικό σύστημα.

Μερικά από τα ειδικά χαρακτηριστικά του BFS περιλαμβάνουν:

- Η δυνατότητα να αντιπροσωπευθούν πολλαπλές συσκευές μέσων ως ενιαίο partition ή volume.
- Υποστήριξη για τις προηγμένες αποθηκευτικές μεθόδους.
- Βελτιστοποιήσεις απόδοσης για τις εφαρμογές πολυμέσων.
- Φορητότητα: Τα partitions BFS μπορούν να κινηθούν μεταξύ των διαφορετικών πλατφόρμων υλικού εύκολα.

Αυτά είναι τα περισσότερα από τα οφέλη που συνδέονται με τα συστήματα αρχείων Unix, αν και επειδή υπάρχουν φυσικά ανταλλαγές δεν θα ήταν ακριβές για να πει κανείς ότι το BFS ήταν "καλύτερο από" τα συστήματα αρχείων Unix.

Το BFS δεν χρησιμοποιείται σε πολλά PCs, για τον απλό λόγο ότι το BeOS δεν εφαρμόζεται ευρέως. Τα λειτουργικά συστήματα WINDOWS δεν μπορούν να έχουν πρόσβαση στα partitions BFS, αλλά κάποιος μπορεί να γράψει έναν οδηγό που θα επιτρέψει στο DOS ή τα Windows για να έχουν πρόσβαση σε BFS. Υπάρχει επίσης ένα interface που επιτρέπει στα partitions BFS να χρησιμοποιηθούν από Linux, το οποίο απεικονίζει ως ένα ορισμένο βαθμό το γεγονός ότι και Linux και BeOS τείνουν να προσελκύσουν την ίδια βάση χρηστών -- τεχνικά πεπειραμένοι χρήστες PC που ψάχνουν κάτι έξω από τη σφαίρα των λειτουργικών συστημάτων της Microsoft.

4.3 – FAT16 – Το σύστημα αρχείων που χρησιμοποιήθηκε

Για την διεκπεραίωση της παρούσης εργασίας χρησιμοποιήθηκε το σύστημα αρχείων FAT16 της οικογένειας FAT. Το σύστημα επιλέχθηκε ανάμεσα στα άλλα για την ευκολία που παρουσιάζει στην δομή του σε σχέση με άλλα συστήματα αρχείων (FAT32,NTFS) ενώ παράλληλα πληρεί όλες τις απαιτήσεις για την δημιουργία ενός εύχρηστου συστήματος. Το σύστημα δεν υποστηρίζει την ανάγνωση μεγάλων ονομάτων κάτι που δεν αποτελεί πρόβλημα καθώς ο τίτλος του τραγουδιού εμφανίζεται μέσω της

πληροφορίας που έχει το ID3 tag και την ύπαρξη υποφακέλων που δεν θεωρήθηκε απαραίτητο για την καλή λειτουργία και δεν υλοποιήθηκε χάριν απλότητας. Παρόλο αυτό υποστηρίζει την σωστή ανάγνωση ενός κατακερματισμένου αρχείου (fragmented) κάτι που δίνει στο χρήστη την δυνατότητα να εναλλάσσει εύκολα τα αρχεία στην κάρτα χωρίς να χρειάζεται να φορμάρει την κάρτα κάθε φορά που επιθυμεί να σβήσει ή να φορτώσει ένα καινούριο τραγούδι. Ακόμα είναι πολύ σημαντικό στοιχείο την επιλογή του FAT16 είναι ότι το μέγεθος της κάρτας υποστηρίζεται από το σύστημα αρχείων.

Πίνακας 4.1: Στοιχεία που υποστηρίζει η οικογένεια FAT

	FAT12	FAT16	FAT32
Used For	Floppies and small hard disk volumes	Small to large hard disk volumes	Medium to very large hard disk volumes
Size of Each FAT Entry	12 bits	16 bits	28 bits
Maximum Volume Size	16 Meg	2 Gig	about 2^41

Παρακάτω γίνεται μία αναλυτική αναφορά του συστήματος αρχείων FAT16.

4.3.1 – FAT16 – Λίγα λόγια ακόμα

Όπως έχει ήδη ειπωθεί όλα τα συστήματα αρχείων FAT κατασκευάστηκαν αρχικά για τον IBM PC. Η σημασία αυτού του γεγονότος είναι ότι το FAT16 είναι “little endian”. Έτσι αν κοιτάξουμε σε μία εγγραφή του FAT16 που καταλαμβάνει 4 bytes θα έχει την εξής μορφή:

```
byte[3]        3 3 2 2 2 2 2 2
               1 0 9 8 7 6 5 4

byte[2]        2 2 2 2 1 1 1 1
               3 2 1 0 9 8 7 6

byte[1]        1 1 1 1 1 1 0 0
               5 4 3 2 1 0 9 8

byte[0]        0 0 0 0 0 0 0 0
               7 6 5 4 3 2 1 0
```

Εικόνα 4.2 : Εικόνα συστήματος “little endian”

Όπως βλέπουμε το byte που συναντάμε πρώτο περιέχει τα μικρότερης αξίας bit ενώ το τελευταίο τα μεγαλύτερης.

4.3.2 – Ο Master Boot Record (MBR)

Όταν ανοίγετε το PC σας, ο επεξεργαστής πρέπει να αρχίσει. Εντούτοις, η μνήμη συστημάτων σας είναι κενή, και ο επεξεργαστής δεν έχει τίποτα για να εκτελέσει, ή στην πραγματικότητα ακόμη να ξέρει που είναι αυτό που πρέπει να εκτελέσει. Για να εξασφαλίσουν ότι το PC μπορεί πάντα να αρχίσει ανεξάρτητα από ποιο BIOS είναι στη μηχανή, οι κατασκευαστές τσιπ και οι κατασκευαστές BIOS κανονίζουν έτσι ώστε ο επεξεργαστής, μόλις ανοιχτεί, να αρχίζει πάντα στην ίδια θέση, FFFF0h.

Κατά τρόπο παρόμοιο, κάθε σκληρός δίσκος πρέπει να έχει μια συνεπή "αφετηρία" όπου οι βασικές πληροφορίες αποθηκεύονται για το δίσκο, όπως πόσα partitions έχουν, ποιο είδος partitions είναι, κλπ.... Επίσης πρέπει να είναι κάπου, ότι το BIOS μπορεί να φορτώσει σαν αρχικό πρόγραμμα έναρξης που αρχίζει τη διαδικασία του λειτουργικού συστήματος. Η θέση όπου αυτές οι πληροφορίες αποθηκεύονται καλείται κύριο αρχείο έναρξης (MBR). Επίσης μερικές φορές καλείται σαν κύριος τομέα έναρξης (Master Boot Sector) ή ακόμα και μόνο τομέα έναρξης. Το κύριο αρχείο έναρξης βρίσκεται πάντα στον κύλινδρο 0, το κεφάλι 0, και τον τομέα 1, ο πρώτος τομέας σε δίσκο και έχει μέγεθος 512 bytes. Ο MBR έχει την παρακάτω μορφή:

Πίνακας 4.2 : Μορφή του MBR

Offset	Description	Size
000h	Executable Code (Boots Computer)	446 Bytes
1BEh	1st Partition Entry (See Next Table)	16 Bytes
1CEh	2nd Partition Entry	16 Bytes
1DEh	3rd Partition Entry	16 Bytes
1EEh	4th Partition Entry	16 Bytes
1FEh	Executable Marker (55h AAh)	2 Bytes

Αυτή είναι η συνεπής "αφετηρία" που ο δίσκος χρησιμοποιεί πάντα. Όταν εκκινεί το BIOS τη μηχανή, αυτή θα κοιτάξει για τις οδηγίες και τις πληροφορίες για το πώς να αρχίσει το δίσκο και να φορτώσει το λειτουργικό σύστημα. Ο MBR περιέχει τις ακόλουθες δομές:

- Κύριος πίνακας partitions: Αυτός ο μικρός πίνακας περιέχει τις περιγραφές των partitions που περιλαμβάνονται στον σκληρό δίσκο. Υπάρχει χώρος στον κύριο πίνακα των partitions για τις πληροφορίες περιγραφής τεσσάρων partitions. Επομένως, ένας σκληρός δίσκος μπορεί να έχει μόνο τέσσερα αληθινά partitions, που καλούνται επίσης αρχικά partitions. Οποιαδήποτε πρόσθετα partitions είναι λογικά partitions που συνδέονται με ένα από τα αρχικά partitions. Ένα από τα partitions είναι χαρακτηρισμένο σαν ενεργό, δείχνοντας ότι είναι αυτό που ο υπολογιστής πρέπει να χρησιμοποιήσει για να εκκινήσει.
- Κύριος κώδικας έναρξης: Το κύριο αρχείο έναρξης περιέχει το μικρό αρχικό πρόγραμμα έναρξης που το BIOS φορτώνει και εκτελεί για να αρχίσει τη διαδικασία έναρξης. Αυτό το πρόγραμμα μεταφέρει τελικά τον έλεγχο στο πρόγραμμα έναρξης που αποθηκεύεται σε οποιοδήποτε partition χρησιμοποιείται για την έναρξη του PC.

Λόγω στη μεγάλη σημασία των πληροφοριών που αποθηκεύονται στο κύριο αρχείο έναρξης, εάν χαλάσει ή αλλοιωθεί με κάποιο τρόπο, η σοβαρή απώλεια στοιχείων μπορεί να

είναι -- στην πραγματικότητα, συχνά θα είναι -- το αποτέλεσμα. Αφού ο κώδικας του MBR είναι το πρώτο πρόγραμμα που εκτελείται όταν ανοίγετε το PC είναι και ο κύριος στόχος των ιών.

Πίνακας 4.3 : Μορφή του partition entry στον MBR

Offset	Description	Size
00h	Current State of Partition (00h=Inactive, 80h=Active)	1 Byte
01h	Beginning of Partition - Head	1 Byte
02h	Beginning of Partition - Cylinder/Sector (See Below)	1 Word
04h	Type of Partition (See List Below)	1 Byte
05h	End of Partition - Head	1 Byte
06h	End of Partition - Cylinder/Sector	1 Word
08h	Number of Sectors Between the MBR and the First Sector in the Partition	1 Double Word
0Ch	Number of Sectors in the Partition	1 Double Word

Ο τύπος του partition μπορεί να είναι ένας από τους παρακάτω (κυριότεροι):

Πίνακας 4.4 : Τύπος partition

Value	Description
00h	Unknown or Nothing
01h	12-bit FAT
04h	16-bit FAT (Partition Smaller than 32MB)
05h	Extended MS-DOS Partition
06h	16-bit FAT (Partition Larger than 32MB)
0Bh	32-bit FAT (Partition Up to 2048GB)
0Ch	Same as 0BH, but uses LBA_1 13h Extensions
0Eh	Same as 06H, but uses LBA_1 13h Extensions
0Fh	Same as 05H, but uses LBA_1 13h Extensions

Η MMC δεν περιέχει MBR και το πρώτο sector της καταλαμβάνεται από το Volume Boot Record. Αυτό συμβαίνει γιατί η MMC έχει μόνο ένα partition.

4.3.3 – Ο Volume Boot Record (VBR)

Όπως ήδη έχει ειπωθεί η κάρτα μνήμης δεν έχει MBR σο πρώτο sector. Αντίθετα εκεί βρίσκεται ο VBR. Αυτός περιέχει σημαντικές πληροφορίες για την δομή του μέσου αποθήκευσης απαραίτητες τόσο για την εύρεση των διάφορων δομών του FAT που χρειάζονται για την προσπέλαση των πληροφοριών του συστήματος όσο και της εύρεσης της περιοχής δεδομένων. Μία εικόνα του VBR φαίνεται παρακάτω:

Πίνακας 4.5 : Δομή του VBR

Offset	Description	Size
00h	Jump Code + NOP	3 Bytes
03h	OEM Name	8 Bytes
0Bh	Bytes Per Sector	1 Word
0Dh	Sectors Per Cluster	1 Byte
0Eh	Reserved Sectors	1 Word
10h	Number of Copies of FAT	1 Byte
11h	Maximum Root Directory Entries	1 Word
13h	Number of Sectors in Partition Smaller than 32MB	1 Word
15h	Media Descriptor (F8h for Hard Disks)	1 Byte
16h	Sectors Per FAT	1 Word
18h	Sectors Per Track	1 Word
1Ah	Number of Heads	1 Word
1Ch	Number of Hidden Sectors in Partition	1 Double Word
20h	Number of Sectors in Partition	1 Double Word
24h	Logical Drive Number of Partition	1 Word
26h	Extended Signature (29h)	1 Byte
27h	Serial Number of Partition	1 Double Word
2Bh	Volume Name of Partition	11 Bytes
36h	FAT Name (FAT16)	8 Bytes
3Eh	Executable Code	448 Bytes
1FEh	Executable Marker (55h AAh)	2 Bytes

4.3.4 – Root Directory Entry

Η περιοχή του δίσκου που περιέχει τα entries των αρχείων που βρίσκονται μέσα στον δίσκο λέγεται Root Directory Entry περιοχή. Αυτή έχει την εξής μορφή στο σύστημα FAT16. Κάθε entry καταλαμβάνει 32 bytes χώρου στο Root Directory έτσι μπορούμε να έχουμε 16 entries σε κάθε τομέα μεγέθους 512 bytes. Το entry έχει την εξής μορφή (FAT16, υποστήριξη μόνο για αρχεία 8.3 (8 bytes όνομα – 3 bytes κατάληξη):

Πίνακας 4.6 : Δομή ενός entry

Offset	Length	Value
0	8 bytes	Name
8	3 bytes	Extension
11	byte	Attribute (00ARSHDV) 0: unused bit A: archive bit, R: read-only bit S: system bit D: directory bit V: volume bit
22	word	Time
24	word	Date
26	word	Cluster (desc. below)
28	dword	File Size

Τα πλέον σημαντικά στοιχεία για την εύρεση του αρχείου μέσα στον δίσκο είναι το πρώτο cluster του αρχείου και το μέγεθος του. Τα entries αποθηκεύονται το ένα μετά το άλλο στην περιοχή του Root Directory. Κάθε entry που είναι αχρησιμοποίητο αρχίζει με το πρώτο byte 0x00 (never used), κάθε entry που έχει διαγραφεί αρχίζει με 0xE5 (ετικέτα διαγραμμένου entry) ενώ κάθε entry που απεικονίζει υπάρχον αρχείο μέσα στον δίσκο αρχίζει με το πρώτο χαρακτήρα ASCII του ονόματός του.

4.3.5 – FAT πίνακες

Αυτή η περιοχή περιέχει την πολύ σημαντική πληροφορία για το επόμενο cluster ενός αρχείου. Από την στιγμή που βρεθεί το πρώτο cluster ενός αρχείου από το Root Directory μπορεί ο χρήστης μέσω του FAT να ανακτήσει με τη σειρά όλα τα clusters που περιέχουν δεδομένα του αρχείου. Όμως τι είναι το cluster; Επειδή είναι πολύ μεγάλο το κόστος σε μνήμη να κρατείται η διεύθυνση κάθε sector στο FAT (ειδικά σε μεγάλους δίσκους) καθώς και ο χρόνος προσπέλασης έπειτα, τα αρχεία οργανώνονται σε clusters. Ανάλογα το μέγεθος του δίσκου είναι και το μέγεθος του cluster (πάντα πολλαπλάσιο του sector). Δεν αντικαθίσταται στο φυσικό επίπεδο το sector με cluster απλά είναι μία λογική οργάνωση 2 ή περισσότερων sectors σ' ένα cluster. Ποιο είναι το trade – off για αυτή την οργάνωση; Ποτέ δεν υπάρχει μέγεθος αρχείου μικρότερο από του cluster. Συνεπώς αν έχουμε FAT16 με μέγεθος cluster 16K (δηλαδή ένα cluster = 32 sectors), κάτι που ισχύει σε δίσκους μεγέθους 513MB - 1 GB και αποθηκεύσουμε ένα αρχείο 1KB τότε θα έχουμε 31KB χώρου σπαταλημένα. Ας σημειώσουμε εδώ ότι τα clusters χρησιμοποιούνται σαν έννοια μόνο για την περιοχή δεδομένων και αυτή αρχίζει πάντα με το cluster 2 (Τα 2 πρώτα είναι δεσμευμένα). Παρακάτω φαίνεται μία σχέση του μεγέθους του cluster σε σχέση με την χωρητικότητα και το σύστημα αρχείων που χρησιμοποιείται.

Πίνακας 4.7 : Μέγεθος cluster σε σχέση με μέγεθος δίσκου και συστήματος αρχείων

Volume size	FAT16 cluster size	FAT32 cluster size	NTFS cluster size
7 MB?16 MB	2 KB	Not supported	512 bytes
17 MB? 32 MB	512 bytes	Not supported	512 bytes
33 MB? 64 MB	1 KB	512 bytes	512 bytes
65 MB? 128 MB	2 KB	1 KB	512 bytes
129 MB? 256 MB	4 KB	2 KB	512 bytes
257 MB? 512 MB	8 KB	4 KB	512 bytes
513 MB? 1 GB	16 KB	4 KB	1 KB

Αυτά τα clusters οργανώνονται σε αυτό που λέγεται αλυσίδα clusters. Αυτό σημαίνει ότι κάθε cluster δείχνει στο επόμενο του. Αυτό γίνεται μέσα στο FAT όπου η κάθε θέση του cluster έχει ένα κωδικό. Αυτός εξηγείται από τον παρακάτω πίνακα.

Πίνακας 4.8 : Κωδικός cluster στο FAT

FAT Code Range	Meaning
0000h	Available Cluster
0002h-FFEFh	Used, Next Cluster in File
FFF0h-FFF6h	Reserved Cluster
FFF7h	BAD Cluster
FFF8h-FFFF	Used, Last Cluster in File

Ας ασχοληθούμε τώρα λίγο με το πώς δουλεύει μία τέτοια αλυσίδα cluster για ένα αρχείο. Ας κοιτάξουμε πρώτα ένα μη κατακερματισμένο αρχείο πως απεικονίζεται στο FAT. Το αρχείο έχει μέγεθος για την περίπτωση μας (MMC 128MB, cluster size 2K, 4 sectors / cluster) 12KB (6 clusters) και αρχίζει από το cluster 2. Το 2 δείχνει στο 3, το 3 δείχνει στο 4 και ούτω καθεξής μέχρι το 7 που δείχνει με τον κωδικό του ότι είναι το τελευταίο cluster στο αρχείο.

Πίνακας 4.9 : Παράδειγμα defragmented αρχείου

Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8
0x00	0x00	0x03	0x04	0x05	0x06	0x07	0xFF	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xFF	0x00

Παρακάτω φαίνεται ένα παράδειγμα ενός κατακερματισμένου αρχείου με μέγεθος 12KB. Τώρα το πρώτο cluster είναι το 2 που δείχνει στο 8 που είναι το δεύτερο cluster στο αρχείο. Η σειρά είναι 2,8,6,7,4 και 5 που ο κωδικός του δείχνει ότι είναι και το τελευταίο cluster στο αρχείο. Το cluster 3 είναι το τέλος κάποιου άλλου αρχείου.

Πίνακας 4.10 : Παράδειγμα fragmented αρχείου

Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8
0x00	0x00	0x08	0xFF	0x05	0xFF	0x07	0x04	0x06
0x00	0x00	0x00	0xFF	0x00	0xFF	0x00	0x00	0x00

4.3.6 – Περιοχή δεδομένων

Η περιοχή δεδομένων περιέχει τα δεδομένα του δίσκου. Είναι οργανωμένη σε clusters όπως έχει ήδη προειπωθεί και η προσπέλαση της γίνεται σύμφωνα με τα περιεχόμενα του FAT.

4.3.7– Μορφή δίσκου φορμαρισμένου σε FAT16

Σύμφωνα με όλα τα προηγούμενα που ειπώθηκαν μπορούμε πλέον να απεικονίσουμε την μορφή ενός drive φορμαρισμένου σε FAT16. Έτσι αυτό θα έχει την παρακάτω μορφή:

Πίνακας 4.11 : Μορφή του FAT16

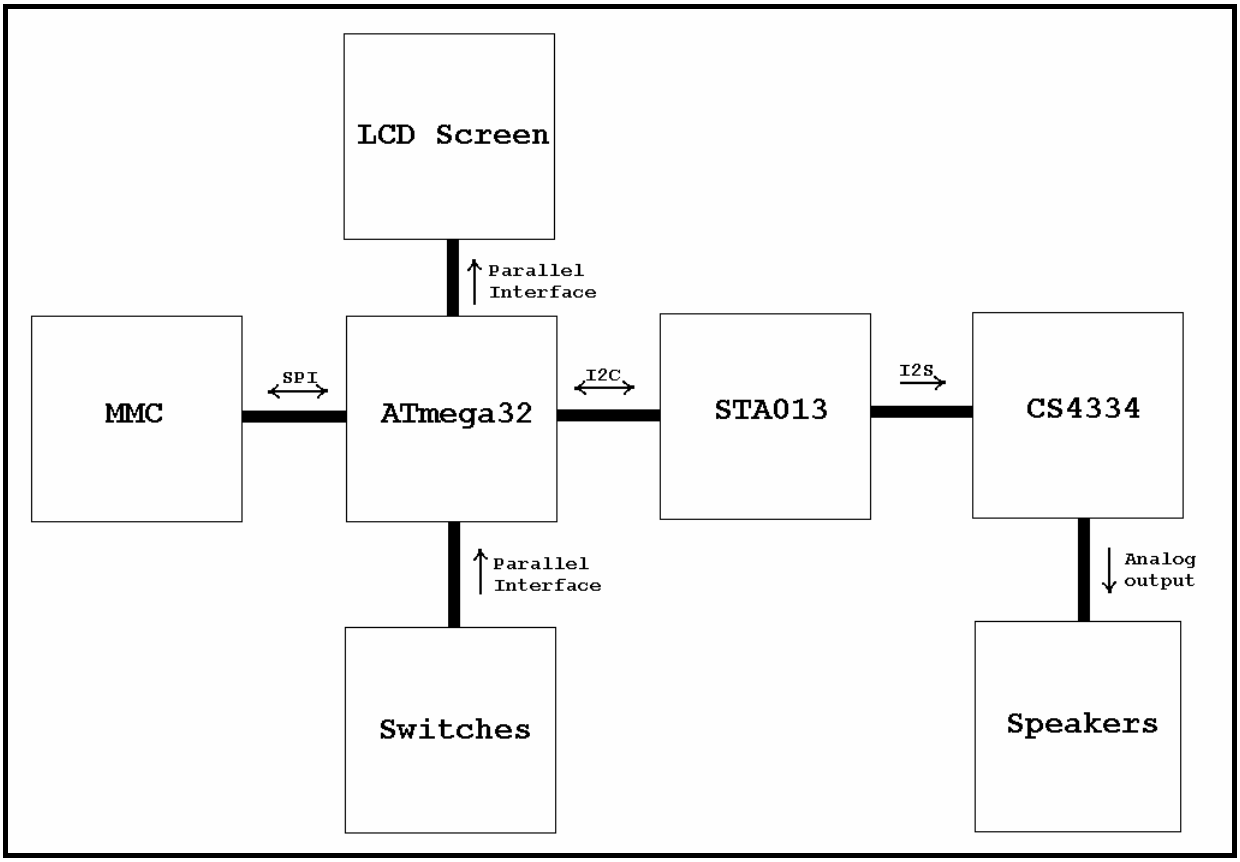
Offset	Description
Start of Partition	Boot Sector
Start + # of Reserved Sectors	Fat Tables
Start + # of Reserved + (# of Sectors Per FAT * 2)	Root Directory Entry
Start + # of Reserved + (# of Sectors Per FAT * 2) + ((Maximum Root Directory Entries * 32) / Bytes per Sector)	Data Area (Starts with Cluster #2)

Κεφάλαιο 5

Η συνδεσμολογία της πλακέτας

5.1– Ο σχεδιασμός του συστήματος

Το σύστημα έχει σχεδιαστεί με την εξής φιλοσοφία: Ένας μικροελεγκτής ελέγχει όλα τα περιφερειακά., ακριβώς η φιλοσοφία που έχουν κατασκευαστεί και όλοι οι μικροελεγκτές. Έτσι η γενική εικόνα του συστήματος σχεδιάστηκε έτσι ώστε ο μικροελεγκτής να διαβάζει τα δεδομένα από την μνήμη του συστήματος και να τα τροφοδοτεί τηρώντας τους σωστούς χρονισμούς στο STA013. Ακόμα ο μικροελεγκτής θα ελέγχει αν πατήθηκε από τον χρήστη κάποιο κουμπί για να εκτελέσει την αντίστοιχη λειτουργία και θα εκτυπώνει στην οθόνη τα επιθυμητά μηνύματα. Έτσι το γενικό πλάνο της πλακέτας θα είναι σαν αυτό που φαίνεται στην παρακάτω εικόνα:

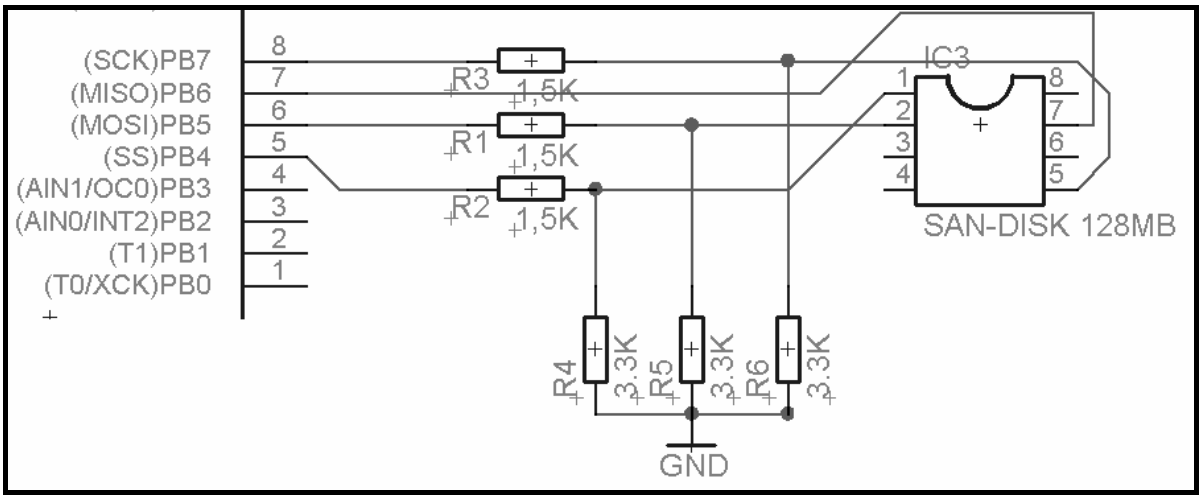


Εικόνα 5.1 : Γενική εικόνα συστήματος

5.2– Η συνδεσμολογία της MMC

Η MMC κάρτα μνήμης συνδέθηκε με το SPI interface του μικροελεγκτή. Η σύνδεση αυτή απαιτεί μετατροπή της τάσεως από 5V σε 3V για τις εισόδους της κάρτας ενώ η έξοδος τη κάρτας (3V) μπορεί να συνδεθεί απευθείας με την είσοδο MISO του AVR καθώς από 2V και ψηλότερα ο AVR αντιλαμβάνεται υψηλό επίπεδο τάσης.

Για αυτή την μετατροπή χρησιμοποιήθηκε ένας καταμεριστής τάσης ο οποίος αποτελείται από αντιστάσεις 1,5 ΚΩ – 2,7 ΚΩ. Παράλληλα με τις αντιστάσεις των 1,5 ΚΩ τοποθετήθηκε πυκνωτής 47 pF για την ταχύτερη ανταπόκριση του κυκλώματος στις γρήγορες αλλαγές των δεδομένων. Το κύκλωμα που χρησιμοποιήθηκε φαίνεται παρακάτω:

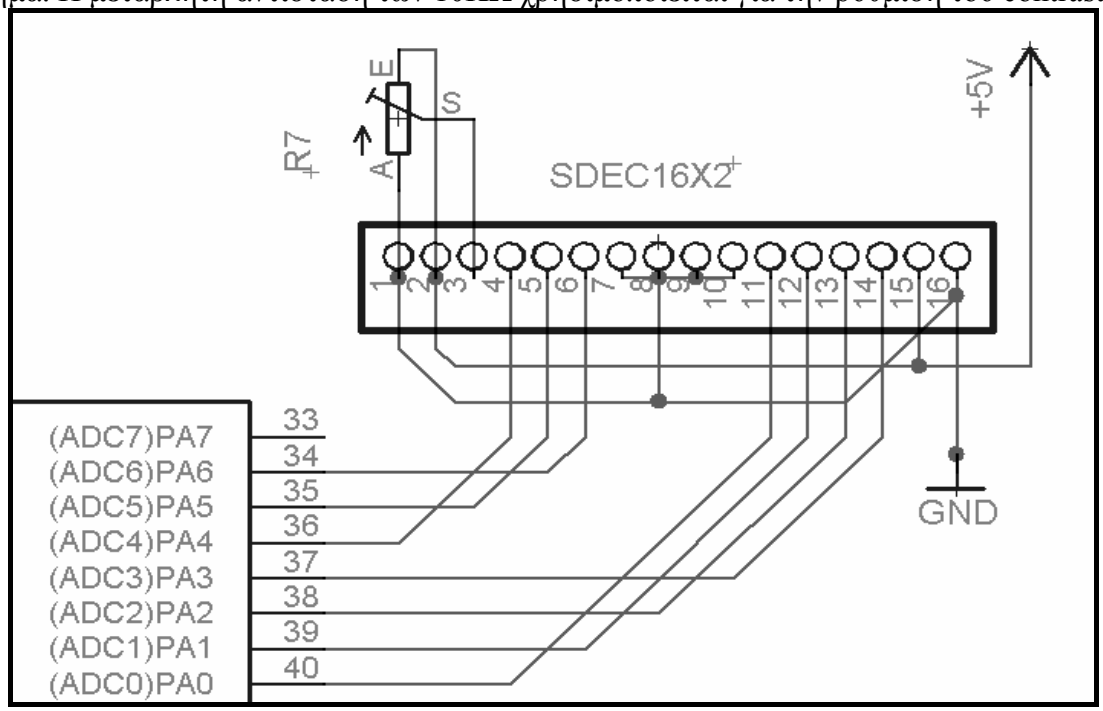


Εικόνα 5.2 : Σύνδεση MMC με AVR

Η συνδεσμολογία του σχήματος λειτούργησε πολύ καλά αφού η τάση στην είσοδο της κάρτας μετρήθηκε στα 3 V ακριβώς ενώ παράλληλα ο χρόνος ανόδου-καθόδου του σήματος ήταν πολύ μικρός σε σχέση με την μέγιστη ταχύτητα που μπορεί να χρησιμοποιηθεί. Καλύτερα ακόμα αποτελέσματα θα μπορούσαν να επιτευχθούν με την χρησιμοποίηση μετατροπέων τάσης (MAXIM, TI κ.α.) οι οποίοι είναι ταχύτεροι αλλά το κόστος και ο χρόνος για την προμήθεια τους θα ανέβαινε με αποτέλεσμα να μην χρησιμοποιηθούν.

5.3– Η συνδεσμολογία της LCD οθόνης

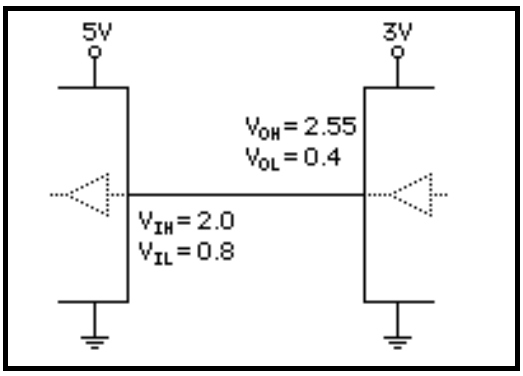
Η οθόνη χαρακτήρων LCD λειτουργεί με 5V και συνδέθηκε σε λειτουργία 4-bit απευθείας στην πόρτα Α του ATmega32. Η συνδεσμολογία που έγινε φαίνεται στο επόμενο σχήμα. Η μεταβλητή αντίσταση των 10KΩ χρησιμοποιείται για την ρύθμιση του contrast.



Εικόνα 5.3 : Σύνδεση LCD με AVR

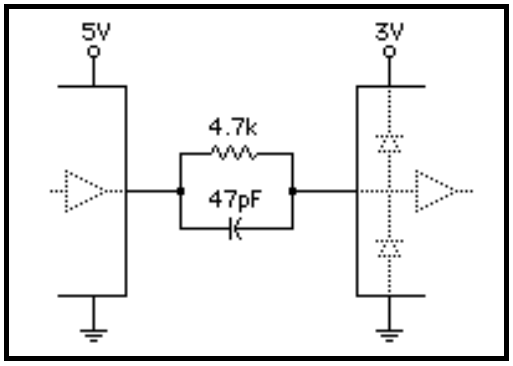
5.4– Η συνδεσμολογία του STA013 και του CS4334

Το STA013 είναι ένα IC 3 V. Οι αξιώσεις των datasheets είναι πως μπορεί να λειτουργήσει μεταξύ 2,7 έως 3,6 V. Δεν πρέπει να χρησιμοποιηθεί σε 5 V. Η δημιουργία μιας παροχής ηλεκτρικού ρεύματος 3 V δεν είναι δύσκολη. Η διασύνδεση του 3 V STA013 στους μικροελεγκτές 5 V και DACs (CS4334) απαιτεί προσοχή για να αποτραπεί η ζημιά στο 3 V STA013. Εάν ένας μικροελεγκτής 5 V πρόκειται να χρησιμοποιηθεί, υπάρχουν μερικοί απλοί τρόποι να συνδεθούν τα τσιπ, εφ' όσον έχουν οι είσοδοι των μικροελεγκτών τα συμβατά κατώτατα όρια εισόδων TTL (σχεδόν όλοι οι μικροελεγκτές είναι έτσι κατασκευασμένοι). Η σύνδεση των εξόδων του STA013 με τις εισόδους επιπέδων TTL είναι απλή, επειδή το STA013 έχει σαν έξοδο τουλάχιστον 85% της παροχής ηλεκτρικού ρεύματος για μια λογική υψηλή, και ένα μέγιστο 0,4 V για μια λογική χαμηλή. Ακόμα κι αν το STA013 τροφοδοτείται από 2,7 V, ακόμα και τότε η έξοδος θα είναι 2,3 V, τα οποία θα ικανοποιήσουν τη 2,0 V απαίτηση μιας εισόδου επιπέδων TTL. Τα τέσσερα σήματα εισόδου του CS4334 είναι όλα TTL.



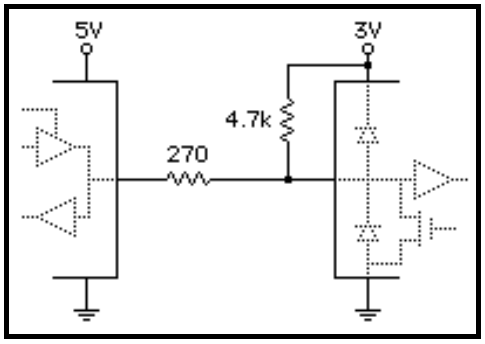
Εικόνα 5.4 : Σύνδεση εξόδου STA013 με AVR και CS4334

Οι έξοδοι από ένα 5 V τροφοδοτούμενο τσιπ δεν πρέπει να συνδεθούν άμεσα με τις εισόδους του STA013. Οι είσοδοι του STA013 δεν μπορούν να δεχθούν 5 V. Κάθε είσοδος έχει ένα ζευγάρι διόδων προστασίας. Αυτές οι δίοδοι μπορούν να διευθύνουν ένα μικρό ρεύμα. Ο απλούστερος και ευκολότερος τρόπος να διασυνδεθεί μια παραγωγή 5 V στις εισόδους του STA013 είναι με έναν αντιστάτη σειράς, ο οποίος θα περιορίσει το ρεύμα όταν η έξοδος 5 V είναι υψηλή. Υπάρχει κάποια χωρητικότητα εισόδου (3,5 pF) στις εισόδους, έτσι η προσθήκη ενός μικρού πυκνωτή παράλληλα με τον αντιστάτη θα επιτρέψει στη γρήγορη ακμή να οδηγήσει κατάλληλα την είσοδο. Η αξία του αντιστάτη και του πυκνωτή δεν είναι κρίσιμες, το σχήμα παρακάτω παρουσιάζει 4.7K και 47pF, που περιορίζει το ρεύμα κατάστασης στο λιγότερο από 1/2 mA. Ο πυκνωτής δεν είναι πραγματικά απαραίτητος εάν η έξοδος δεν πρέπει να οδηγηθεί γρήγορα, όπως η είσοδος RESET.



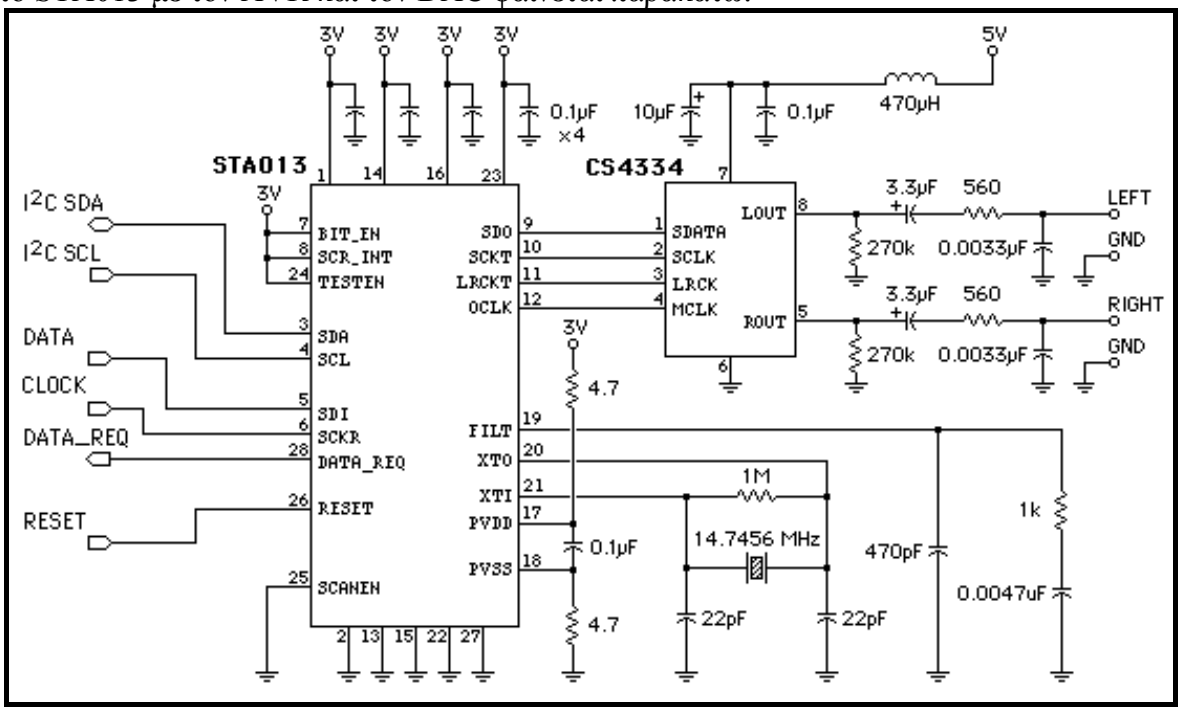
Εικόνα 5.5 : Σύνδεση εισόδου STA013 με AVR

Η σύνδεση του σήματος SDA από έναν μικροελεγκτή 5 V με το STA013 είναι λίγο πιό σύνθετη, αν και ένα απλό κύκλωμα μπορεί συχνά να χρησιμοποιηθεί. Η γραμμή SDA είναι αμφίδρομη, όπου καθεμία συσκευή μπορεί να «τραβήξει κάτω», και ένας αντιστάτης παρέχει pull-up. Οι περισσότεροι μικροελεγκτές έχουν τα κατώτατα όρια TTL, έτσι ένα pull-up στον ανεφοδιασμό 3 V θα ικανοποιήσει εύκολα τη 2,0 V ελάχιστη υψηλή απαίτηση. Εάν ο μικροελεγκτής δεν θα οδηγήσει ποτέ τη γραμμή υψηλή (μόνο τράβηγμα χαμηλά ή tri-stated), κατόπιν κανένα άλλο μέρος δεν απαιτείται πιθανώς. Αυτό μπορεί να συμβεί εάν ο μικροελεγκτής έχει μια αφιερωμένη έξοδο I²C. Στις περισσότερες περιπτώσεις, ο μικροελεγκτής μπορεί να οδηγήσει τη γραμμή υψηλά, και ένας πρόσθετος αντιστάτης πρέπει να προστεθεί για να αποτρέψει τη ζημιά στο STA013. Αυτός ο τρέχων αντιστάτης περιορισμού πρέπει να είναι μικρός, δεδομένου ότι θα διαμορφώσει έναν διαιρέτη αντιστατών με το pull-up όταν οδηγεί χαμηλά ο μικροελεγκτής.



Εικόνα 5.6 : Σύνδεση αμφίδρομης γραμμής SDA

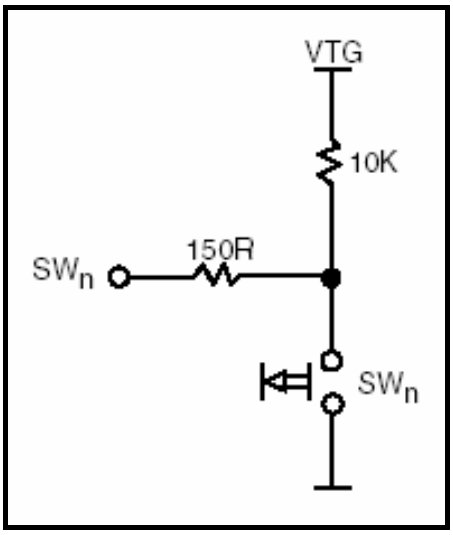
Η σύνδεση που έγινε για την δημιουργία του κυκλώματος φαίνεται στα παραπάνω σχήματα. Η Phillips προτείνει μία πιο περίπλοκη σύνδεση (αναφέρεται σε προηγούμενο κεφάλαιο) η οποία είναι βασισμένη σε transistor και είναι απαραίτητη σε κυκλώματα που έχουν περισσότερες από μία συσκευές στο I^2C διάδρομο. Τελικά το κύκλωμα που συνδέει το STA013 με τον AVR και τον DAC φαίνεται παρακάτω.



Εικόνα 5.7 : Συνδεσμολογία STA013 και CS4334

5.5– Η συνδεσμολογία των dip-switches

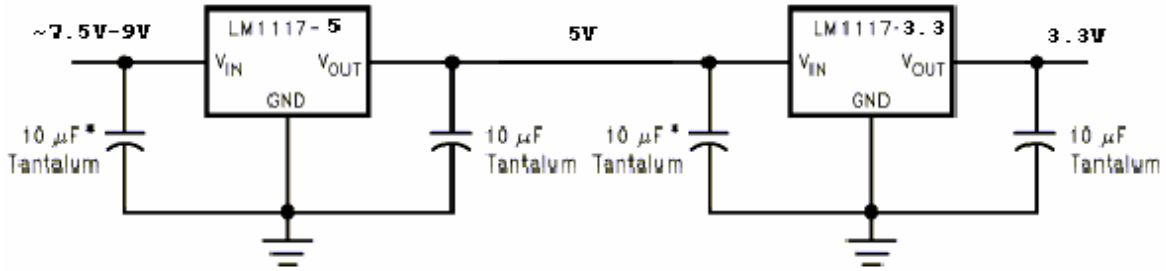
Η συνδεσμολογία που ακολουθήθηκε για την σύνδεση των dip-switches είναι ίδια με αυτή στο αναπτυξιακό STK500.



Εικόνα 5.8 : Συνδεσμολογία dip-switch με είσοδο AVR

5.6– Η τροφοδοσία και η έξοδος της πλακέτας

Η τροφοδοσία της πλακέτας έγινε μέσω δύο voltage regulator από ένα μετασχηματιστή. Η συνδεσμολογία με τους voltage regulators χρησιμοποιήθηκε για να παραχθούν οι δύο απαιτούμενες τάσεις τροφοδοσίας (3V και 5V) ενώ παράλληλα η χρησιμοποίηση των voltage regulators μας δίνει την δυνατότητα να χρησιμοποιήσουμε μετασχηματιστή αντί για γεννήτρια. Η συνδεσμολογία των voltage regulators φαίνεται παρακάτω:



Εικόνα 5.9 : Συνδεσμολογία των voltage regulators

Η έξοδος του DAC οδηγείται κατευθείαν σε ηχεία αφού μπορεί να οδηγήσει ηχεία αλλά και ακουστικά.

Κεφάλαιο 6

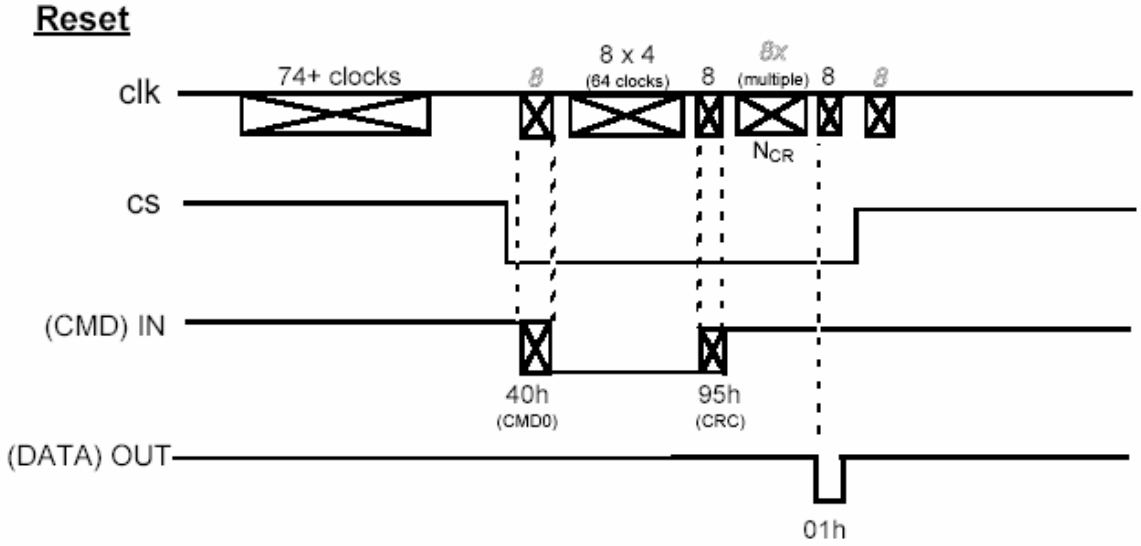
Ανάλυση του προγράμματος

6.1 – Περιγραφή προγράμματος (driver) για την κάρτα

Όπως έχει διατυπωθεί και προηγουμένως για την λειτουργία της κάρτας χρησιμοποιήθηκε το SPI interface. Για την οδήγηση της κάρτας γράφτηκε πρόγραμμα το οποίο εκπληρώνει τις εξής βασικές συναρτήσεις για την κάρτα: επανεκκίνηση, αρχικοποίηση, ορισμός παραμέτρων της κάρτας για την μετέπειτα επικοινωνία, ανάγνωση ενός sector (512 bytes) και εγγραφή ενός sector (512 bytes). Επίσης χρησιμοποιήθηκαν και οι συναρτήσεις αρχικοποίησης του SPI και μετάδοσης ενός byte με το SPI οι οποίες περιγράφονται στην επεξήγηση του μικροελεγκτή σε προηγούμενο κεφάλαιο. Αξιοσημείωτο είναι οι παλμοί ρολογιού που απαιτούνται για να ληφθεί η απάντηση από την κάρτα και φαίνονται στα σχηματικά διαγράμματα. Ο πηγαίος κώδικας σε γλώσσα C για τον compiler Codevision AVR βρίσκεται στο παράρτημα.

6.1.1 –Συνάρτηση *MMC_Reset* – Επανεκκίνηση της κάρτας

Η κάρτα μνήμης εκτός από το κύκλο τροφοδότησης που μπορεί να γίνει με φυσικό τρόπο (διακοπή της τροφοδότησης για λίγο) μπορεί να επανεκκινηθεί και με ειδική εντολή. Για αυτή την εντολή ακολουθήθηκε η εξής διαδικασία. Μετά την κλήση της δημιουργήθηκε καθυστέρηση 250 ms για ασφάλεια έτσι ώστε η τροφοδότηση της κάρτας να φτάσει τα επιθυμητά επίπεδα. Έπειτα ακολούθησε η αποστολή 80 κύκλων ρολογιού με το σήμα επιλογής της κάρτας απενεργοποιημένο τα οποία αναφέρονται ως απαραίτητη λειτουργία στις προδιαγραφές της κάρτας. Έπειτα επιλέγεται η κάρτα και στέλνεται επανειλημμένα η εντολή 0 μέχρι η κάρτα να δώσει την απάντηση 01h που σημαίνει ότι η κάρτα βρίσκεται σε idle κατάσταση και περιμένει την εντολή 1. Ένα σχηματικό διάγραμμα που περιγράφει την εξέλιξη της εντολής φαίνεται παρακάτω:

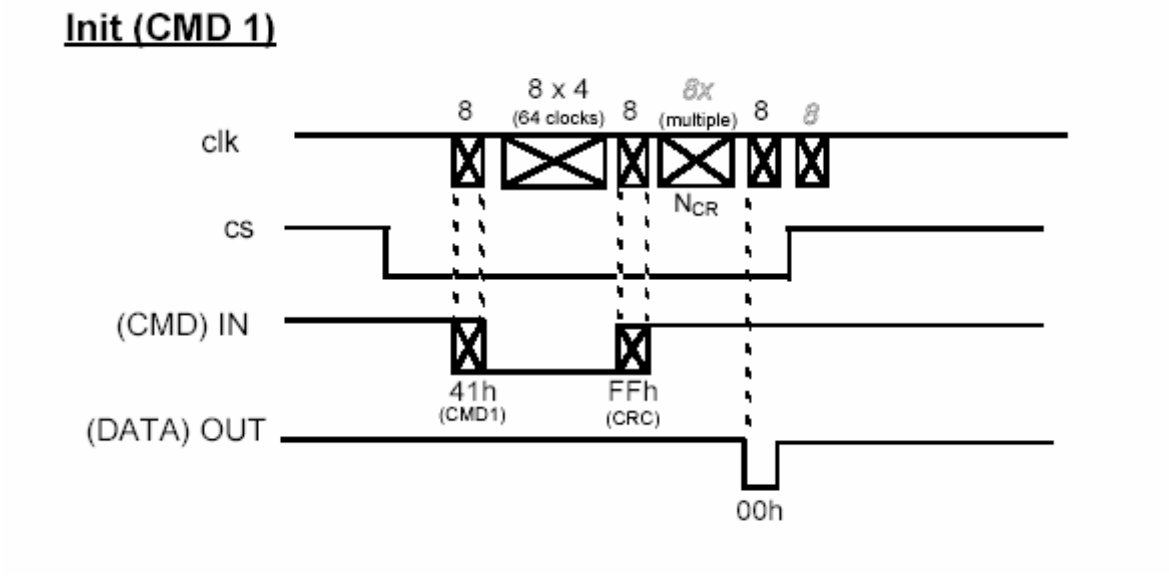


Εικόνα 6.1 : Διαδικασία επανεκκίνησης της MMC με την εντολή 0

6.1.2 –Συνάρτηση MMC_Init – Αρχικοποίηση της κάρτας

Για την αρχικοποίηση της κάρτας χρησιμοποιείται η εντολή 1. Αυτή η εντολή ελέγχει αν η κάρτα εκτελεί ακόμα εσωτερικές λειτουργίες ή έχει τελειώσει με την

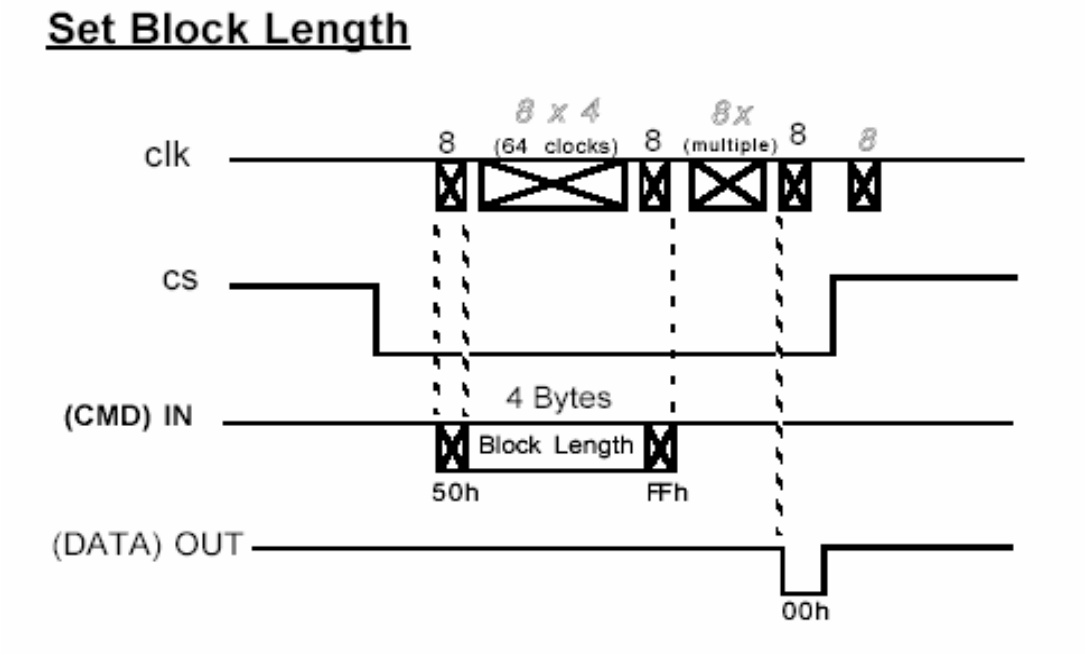
επανεκκίνηση και είναι έτοιμη να δεχτεί εντολές. Η συνάρτηση καλείται αρκετές φορές μέχρι να επιτύχει η αρχικοποίηση. Ένα σχηματικό διάγραμμα που περιγράφει την εξέλιξη της εντολής φαίνεται παρακάτω:



Εικόνα 6.2 : Διαδικασία αρχικοποίησης της MMC με την εντολή 1

6.1.3 –Συνάρτηση *MMC_Set* – Ορισμός παραμέτρων

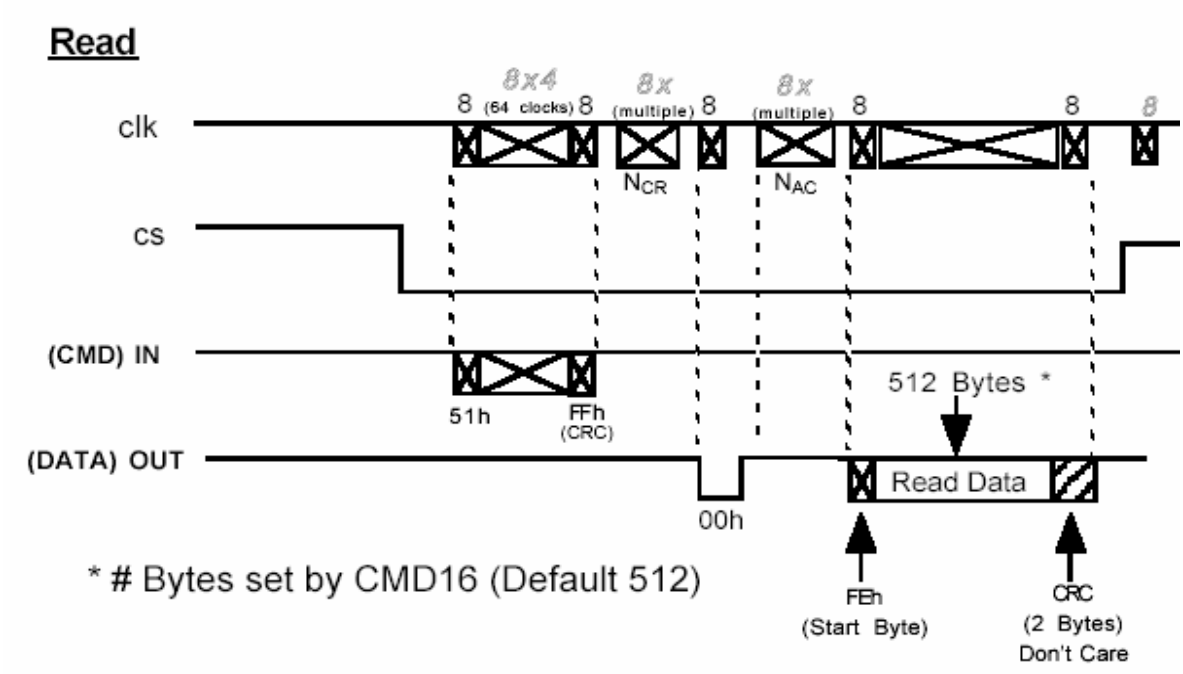
Αυτή η συνάρτηση χρησιμοποιείται αμέσως μετά την αρχικοποίηση για την απενεργοποίηση του ελέγχου CRC και τον ορισμό του μεγέθους σε bytes που θα διαβάζονται και θα εγγράφονται κάθε φορά στην κάρτα. Ο CRC έλεγχος δεν χρησιμοποιείται στην SPI λειτουργία και έτσι απενεργοποιείται με την εντολή 59. Το μέγεθος του block που θα χρησιμοποιείται στις μεταφορές δεδομένων ορίστηκε στα 512 bytes με την εντολή 16. Ένα σχηματικό διάγραμμα που περιγράφει την εξέλιξη της εντολής 16 φαίνεται παρακάτω:



Εικόνα 6.3 : Διαδικασία ορισμού του μεγέθους του block της MMC με την εντολή 16

6.1.4 –Συνάρτηση *MMC_Read Sector* – Ανάγνωση ενός sector

Η συνάρτηση αυτή χρησιμοποιείται για να επιστρέψει τα περιεχόμενα του sector που ζητείται σε ένα buffer 512 θέσεων. Αρχικά στέλνεται η εντολή 17 και λαμβάνεται η απάντηση 00h. Αυτό σημαίνει την σωστή αποδοχή της εντολής από την κάρτα. Έπειτα στέλνονται συνεχώς ρολόγια μέχρι η κάρτα να σημάνει την αρχή του sector με το κατάλληλο data token. Έπειτα δίνονται στην κάρτα 512x 8 παλμοί για να επιστρέψει τα περιεχόμενα του sector που ζητήθηκαν. Στο τέλος στέλνονται ακόμα 16 παλμοί ρολογιού για να τελειώσει η κάρτα τις εσωτερικές της λειτουργίες. Ένα σχηματικό διάγραμμα που περιγράφει την εξέλιξη της εντολής 17 φαίνεται παρακάτω:



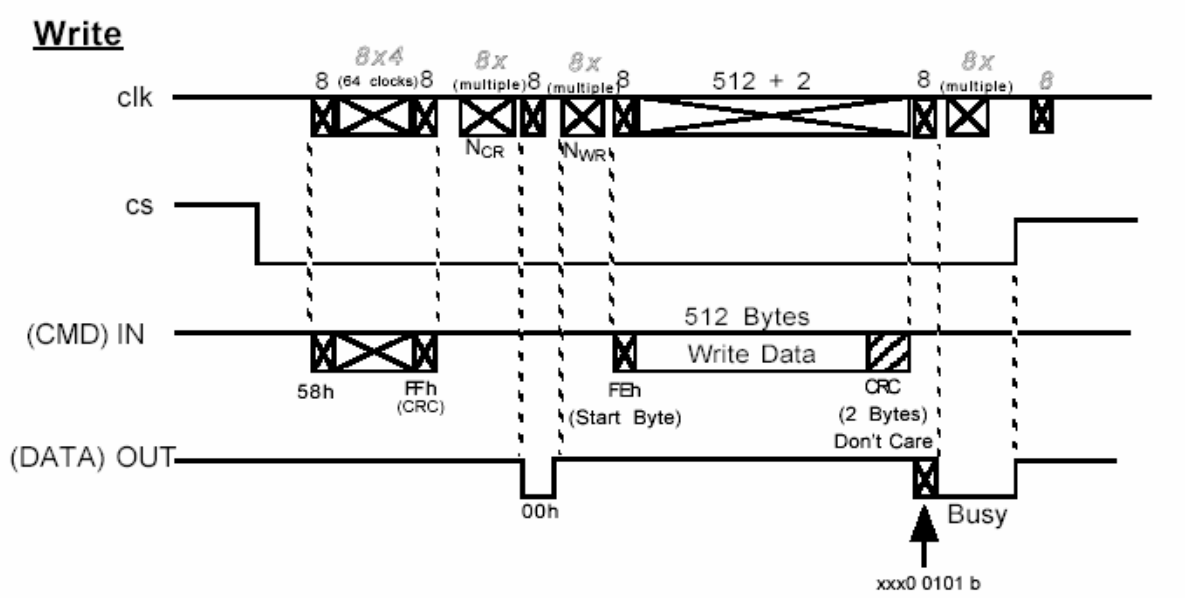
Εικόνα 6.4 : Διαδικασία ανάγνωσης sector από την MMC με την εντολή 17

6.1.5 –Συνάρτηση *MMC_Write Sector* – Εγγραφή ενός sector

Η συνάρτηση αυτή χρησιμοποιείται για την εγγραφή ενός sector της κάρτας (δεν χρησιμοποιείται στο τελικό πρόγραμμα αλλά χρησιμοποιήθηκε για την απασφαλμάτωση του προγράμματος οδήγησης της κάρτας). Αρχικά στέλνεται η εντολή 24 που λέει στην κάρτα ότι πρόκειται να εγγράψει ένα sector, μαζί με την διεύθυνση του sector. Αν η απάντηση της κάρτας είναι επιτυχής τότε δηλώνεται η αρχή της εγγραφής με το κατάλληλο data token και αρχίζουν να εγγράφονται τα bytes ένα προς ένα. Έπειτα ελέγχεται η απάντηση της κάρτας αν παρέλαβε σωστά τα δεδομένα. Ένα σχηματικό διάγραμμα που περιγράφει την εξέλιξη της εντολής 24 φαίνεται παρακάτω:

6.1.6 –Συνάρτηση *MMC_Send Command* – Αποστολή εντολής και ορισμάτων της

Η συνάρτηση αυτή χρησιμοποιείται για την αποστολή εντολών στην κάρτα καθώς και των ορισμάτων τους (καλείται εσωτερικά από τον driver της κάρτας). Αρχικά στέλνεται ένα dummy byte και έπειτα στέλνονται κατά σειρά: ο κωδικός της εντολής με το $7^{ο}$ bit ίσο με 1 για να δηλώνει στην κάρτα ότι είναι εντολή, το όρισμα σε 4 μεταφορές και ο έλεγχος CRC (καθορισμένη τιμή 0x95). Στην συνέχεια δίνονται παλμοί ρολογιού στην κάρτα να επιστρέψει την απάντηση της.



Εικόνα 6.5 : Διαδικασία εγγραφής sector στην MMC με την εντολή 24

6.2 – Περιγραφή driver για την LCD οθόνη

Παρακάτω περιγράφονται αναλυτικά οι συναρτήσεις που δημιουργήθηκαν για την οδήγηση της οθόνης.

6.2.1 – Συνάρτηση LCD Init – Αρχικοποίηση της οθόνης

Η οθόνη απαιτεί ειδική αρχικοποίηση για να δουλέψει σε 4 – bit mode. Αυτή η αρχικοποίηση αποτελείται από τα εξής βήματα:

- Καθυστέρηση 15 ms (τουλάχιστον)
- Αποστολή της εντολής 0x22
- Καθυστέρηση 5 ms (τουλάχιστον)
- Αποστολή της εντολής 0x22
- Αποστολή της εντολής 0x28 – Η οθόνη έχει 2 γραμμές
- Αποστολή της εντολής 0x0C – Ανάβει την οθόνη και σβήνει τον κέρσορα
- Αποστολή της εντολής 0x01 – Καθαρίζει την οθόνη
- Αποστολή της εντολής 0x06 – Καθορίζει την διεύθυνση κίνησης του κέρσορα και ενεργοποιεί την οθόνη

6.2.2 – Συνάρτηση LCD Write – Εγγραφή στην οθόνη

Αυτή η συνάρτηση δημιουργήθηκε για την εγγραφή στην οθόνη τόσο των εντολών όσο και των δεδομένων. Η συνάρτηση δουλεύει ως εξής:

- Το σήμα R/W γίνεται λογικά χαμηλό για να καταλάβει η οθόνη ότι πρόκειται να γίνει εγγραφή.
- Το σήμα RS γίνεται λογικά χαμηλό αν πρόκειται να μεταδοθεί εντολή ενώ λογικά υψηλό αν πρόκειται να μεταδοθούν δεδομένα για απεικόνιση
- Γράφονται τα 4 MSB της εντολής (ή των δεδομένων) πρώτα και έπειτα τα 4 LSB.
- Ελέγχεται η σημαία που δηλώνει ότι η LCD είναι busy μέχρι να γίνει 0.

6.2.3 – Συνάρτηση LCD Read – Ανάγνωση από την οθόνη

Η συνάρτηση αυτή διαβάζει από την οθόνη. Ακολουθείται η ίδια διαδικασία όπως και στην συνάρτηση εγγραφής με την διαφορά ότι το σήμα R/W γίνεται 1 για να προστάξει την οθόνη ότι πρόκειται να διαβαστεί.

6.2.4 – Συνάρτηση LCD Set Position – Καθορισμός θέσης εγγραφής

Η συνάρτηση αυτή χρησιμοποιείται για να καθοριστεί η επόμενη θέση που θα εγγραφεί ο χαρακτήρας πάνω στην οθόνη. Αυτό γίνεται με την εντολή (0x80 + pos) για την $1^{η}$ γραμμή της οθόνης και (0xC0 + pos) για την δεύτερη γραμμή.

6.2.5 – Συνάρτηση LCD Clear – Εκκαθάριση της οθόνης

Αυτή η συνάρτηση στέλνε την εντολή 0x01 για να προστάξει την οθόνη να καθαρίσει όλα τα δεδομένα. Ακόμα μετά καθορίζεται η επόμενη διεύθυνση εγγραφής στον πρώτο χαρακτήρα της πρώτης γραμμής.

6.2.6 – Συνάρτηση LCD Shift – Shift των δεδομένων

Αυτή η συνάρτηση δημιουργήθηκε για να γίνονται shift τα δεδομένα της οθόνης είτε δεξιά είτε αριστερά ανάλογα με την επιθυμία του χρήστη. Αυτό γίνεται με την εντολή 0x1C για δεξιά ολίσθηση και 0x18 για αριστερή ολίσθηση.

6.2.7 – Συνάρτηση LCD Print Char – Εκτύπωση χαρακτήρων

Αυτή η συνάρτηση εγγράφει στην οθόνη τον χαρακτήρα που περνιέται ως παράμετρος από το πρόγραμμα.

6.2.8 – Συνάρτηση LCD Print String – Εκτύπωση String

Αυτή η συνάρτηση παίρνει σαν όρισμα ένα πίνακα χαρακτήρων και χρησιμοποιώντας την προηγούμενη συνάρτηση εκτυπώνει στην οθόνη το αλφαριθμητικό.

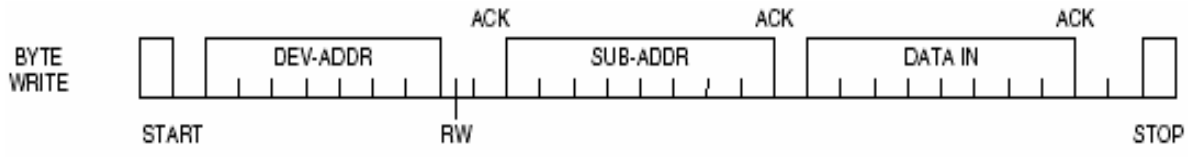
6.3 – Περιγραφή driver για το STA013

Το IC που κάνει την αποκωδικοποίηση των αρχείων .mp3 χρειάζεται το I^2C για να επικοινωνήσει και να ελεγχθεί. Αυτό το βήμα δεν μπορεί να αποφευχθεί. Οι παραπάνω βασικές λειτουργίες του I^2C διαδρόμου χρησιμοποιούνται για να κατασκευαστούν οι βασικές λειτουργίες επικοινωνίας με το STA013.

6.3.1 – Εγγραφή στο STA013

Οι βασικές λειτουργίες του I^2C χρησιμοποιούνται για την δημιουργία μίας ακολουθίας για την εγγραφή στο STA013. Η ακολουθία αυτή είναι η παρακάτω:

1. **Start Condition:** Αλλαγή της γραμμής SDA από 1 σε 0 όταν η γραμμή SCL είναι υψηλή
2. **Send Address Byte:** Η διεύθυνση που ανταποκρίνεται το STA013 είναι η δυαδική b1000011x. Τώρα το τελευταίο bit γίνεται 0 κάτι που λέει στο STA013 ότι θα γράψει.
3. **Send Sub-Address Byte:** Τώρα στέλνεται η εσωτερική διεύθυνση του STA013 που θα εγγραφεί.
4. **Send Value to be written:** Έπειτα στέλνεται η τιμή που θα εγγραφεί.
5. **Stop Condition:** Αλλαγή της γραμμής SDA από 0 σε 1 όταν η γραμμή SCL είναι υψηλή. Αυτό σημάνει το τέλος της επικοινωνίας στον διάδρομο.



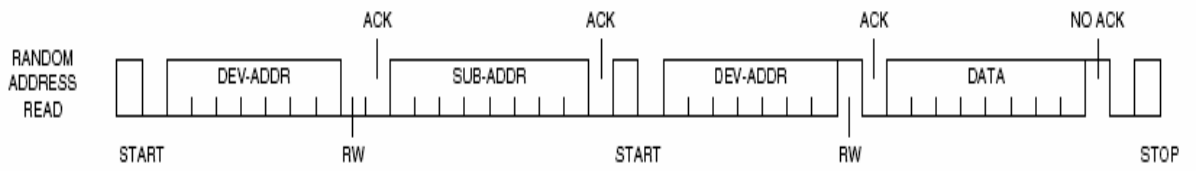
Εικόνα 6.6 : Συνάρτηση εγγραφής στο STA013

6.3.2 – Ανάγνωση από το STA013

Η ανάγνωση από το STA013 είναι λίγο πιο περίπλοκη και γίνεται με τα εξής βήματα:

1. **Start Condition:** Αλλαγή της γραμμής SDA από 1 σε 0 όταν η γραμμή SCL είναι υψηλή
2. **Send Address Byte:** Η διεύθυνση που ανταποκρίνεται το STA013 είναι η δυαδική b1000011x. Τώρα το τελευταίο bit γίνεται 0 κάτι που λέει στο STA013 ότι θα γράψει.
3. **Send Sub-Address Byte:** Τώρα στέλνεται η εσωτερική διεύθυνση του STA013 που θα αναγνωστεί.
4. **Stop Condition:** Αλλαγή της γραμμής SDA από 0 σε 1 όταν η γραμμή SCL είναι υψηλή. Αυτό σημάνει το τέλος της επικοινωνίας στον διάδρομο.
5. **Start Condition:** Αλλαγή της γραμμής SDA από 1 σε 0 όταν η γραμμή SCL είναι υψηλή
6. **Send Address Byte:** Η διεύθυνση που ανταποκρίνεται το STA013 είναι η δυαδική b1000011x. Τώρα το τελευταίο bit γίνεται 1 κάτι που λέει στο STA013 ότι θα διαβάσει.
7. **Read Data from STA013:** Οκτώ παλμοί ρολογιού στέλνονται από τον μικροελεγκτή προς το STA013 για να παραληφθεί το byte των δεδομένων. Έπειτα ο μικροελεγκτής δίνει ACK bit.

8. **Stop Condition:** Αλλαγή της γραμμής SDA από 0 σε 1 όταν η γραμμή SCL είναι υψηλή. Αυτό σημάνει το τέλος της επικοινωνίας στον διάδρομο.



Εικόνα 6.7 : Συνάρτηση ανάγνωσης από το STA013

6.3.3 – Συνάρτηση STA013 HW Reset

Αυτή η συνάρτηση χρησιμοποιείται για να επανεκκινεί το STA013. Απλώς χαμηλώνει την γραμμή Reset και έπειτα την ξαναθέτει σε υψηλό επίπεδο.

6.3.4 – Συνάρτηση STA013 Update

Η συνάρτηση αυτή στέλνει το update αρχείο που παρέχει η ST για το IC. Αυτό το αρχείο περιέχει 2007 εγγραφές στο ολοκληρωμένο και είναι απαραίτητο για την σωστή λειτουργία του.

6.3.5 – Συνάρτηση STA013 Init

Η συνάρτηση αυτή κατ' αρχάς διαβάζει από την διεύθυνση 0x01 του STA013 και ελέγχει αν η απάντηση είναι σωστή (0xAC). Έπειτα καλεί την συνάρτηση STA013 Update για να εγγραφεί το update αρχείο στο STA013.

6.3.6 – Συνάρτηση STA013 Start

Η συνάρτηση αυτή καλείται για να περάσει τις απαραίτητες ρυθμίσεις στο STA013 ώστε η έξοδος του να είναι σε μορφή συμβατή με τον DAC CS4334. Στο τέλος στέλνονται στο STA013 οι εντολές RUN και PLAY.

6.3.7 – Συνάρτηση STA013 Run

Η συνάρτηση αυτή διατάσει το STA013 να αρχίσει να λειτουργεί εγγράφοντας στον RUN καταχωρητή την τιμή 0x01.

6.3.8 – Συνάρτηση STA013 Play

Η συνάρτηση αυτή διατάσει το STA013 να αρχίσει να λειτουργεί εγγράφοντας στον Play καταχωρητή την τιμή 0x01.

6.3.9 – Συνάρτηση STA013 Pause

Η συνάρτηση αυτή διατάσει το STA013 να σταματήσει να λειτουργεί εγγράφοντας στον Play καταχωρητή την τιμή 0x00.

6.3.10 – Συνάρτηση STA013 Mute

Η συνάρτηση αυτή διατάσει το STA013 να σταματήσει να παράγει ήχο εγγράφοντας στον Mute καταχωρητή την τιμή 0x01 όταν παίρνει σαν όρισμα TRUE. Ακόμα όταν παίρνει σαν όρισμα FALSE θέτει σε DE-MUTE το STA013.

6.3.11 – Συνάρτηση STA013 Volume

Η συνάρτηση αυτή ορίζει την ένταση της φωνής στο STA013 εγγράφοντας την τιμή της έντασης στους κατάλληλους καταχωρητές.

6.3.12 – Συνάρτηση STA013 Send Data

Η συνάρτηση αυτή στέλνει τα δεδομένα στο STA013. Η συνάρτηση στέλνει κάθε bit (με το MSB πρώτα) στην θετική ακμή του ρολογιού. Για την δημιουργία της χρησιμοποιήθηκε η τεχνική loop – unrolling η οποία επιταχύνει το πρόγραμμα σε 11 κύκλους μηχανής / bit. Περαιτέρω επιτάχυνση θα μπορούσε να επιτευχθεί με την εγγραφή του προγράμματος σε assembly.

6.4 – Περιγραφή driver για το FAT16

Παρακάτω αναφέρονται οι συναρτήσεις που δημιουργήθηκαν για την ανάγνωση του FAT16 συστήματος της MMC.

6.4.1 – Συνάρτηση FAT Read BootRecord

Η συνάρτηση αυτή διαβάζει το πρώτο sector της MMC και κρατά στην RAM τα στοιχεία που ενδιαφέρουν. Επίσης υπολογίζει τις αρχικές διευθύνσεις των τεσσάρων βασικών περιοχών του συστήματος αρχείων.

6.4.2 – Συνάρτηση Clust2Sec

Η συνάρτηση αυτή μετατρέπει τον αριθμό του cluster σε φυσική διεύθυνση της MMC (αριθμό sector).

6.4.3 – Συνάρτηση Get Next Clust Add

Η συνάρτηση αυτή βρίσκει το ίχνος του τρέχοντος cluster μέσα στο FAT και αποθηκεύει την τιμή του αφού αυτό το ίχνος είναι η διεύθυνση του επόμενου cluster στην αλυσίδα.

6.4.4 – Συνάρτηση Get Next Sector

Η συνάρτηση αυτή διαβάζει το επόμενο sector από το cluster που είναι τρέχον. Αν το cluster είναι το τελευταίο στην αλυσίδα τότε καλεί την Get Next File που βρίσκει το 1^{ο} cluster του επόμενου αρχείου.

6.4.5 – Συνάρτηση Get Next File

Η συνάρτηση διατρέχει το Root Directory Table και βρίσκει την επόμενη έγκυρη εγγραφή. Έπειτα από αυτό ανανεώνει την διεύθυνση του current cluster.

6.4.6 – Συνάρτηση Get Previous File

Η συνάρτηση αυτή διατρέχει προς τα πίσω το Root Directory Table και βρίσκει το αμέσως προηγούμενο αρχείο.

6.4.7 – Συνάρτηση Show Tag Info

Η συνάρτηση Show Tag Info εκτυπώνει στην LCD οθόνη την Tag πληροφορία του τραγουδιού. Η Tag πληροφορία είναι έξτρα πληροφορία στο τέλος του τραγουδιού για τον τίτλο του, το όνομα του τραγουδιστή και άλλα στοιχεία για το mp3.

6.4.7.1 – Λίγα λόγια για το ID3 tag

Η ID3 tag είναι 128 bytes στο τέλος του τραγουδιού. Έχει το παρακάτω format:

Πίνακας 6.1 : Μορφή ID3 Tag

Song title	30 characters
Artist	30 characters
Album	30 characters
Year	4 characters
Comment	30 characters
Genre	1 byte

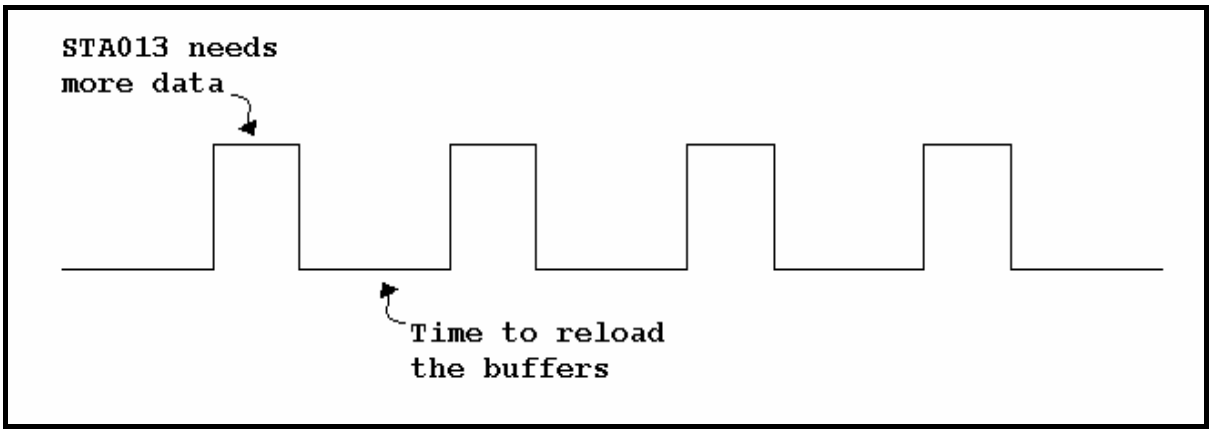
Αν αθροίσει κανείς θα δει ότι τα παραπάνω είναι 125 bytes και όχι 128. Τα υπόλοιπα τρία είναι η λέξη “TAG”. Εκεί στηρίζεται η συνάρτηση για να βρει το tag info και να το απεικονίσει στην οθόνη. Αρχικά ψάχνει στο FAT για το τελευταίο cluster του τραγουδιού. Αυτό το cluster πάντα περιέχει το TAG information. Έπειτα αφού βρει το τελευταίο cluster το πρόγραμμα ψάχνει μέσα στα sectors αυτού για την λέξη “TAG”. Μόλις βρεθεί η λέξη αποθηκεύονται τα επόμενα 30 στοιχεία που είναι το όνομα του τραγουδιού και τα επόμενα 30 που είναι το όνομα του τραγουδιστή. Έπειτα τα δύο αλφαριθμητικά εκτυπώνονται στις δύο γραμμές της οθόνης.

6.5 – Key Debouncing

Για την εργασία χρησιμοποιήθηκαν dip – switches. Για τον έλεγχο τους χρησιμοποιήθηκε ο timer 0. Κάθε 50 ms (περίπου) ο timer 0 ρυθμίστηκε να χτυπάει διακοπή. Τότε ελέγχονται τα κουμπιά για πάτημα. Αυτή η περιοδική μέθοδος ελέγχου των κουμπιών σώζει χρόνο εκτέλεσης που θα έπρεπε να σπαταληθεί αν περίμενε το πρόγραμμα για θετική ακμή, αφού έπειτα θα έπρεπε να περιμένει για τον debouncing χρόνο.

6.6 – Κυρίως πρόγραμμα

Το κυρίως πρόγραμμα πρέπει να φροντίζει κυρίως για την σωστή και έγκυρη τροφοδότηση του αποκωδικοποιητή των mp3 με δεδομένα. Για την επίτευξη αυτού του σκοπού το πρόγραμμα στηρίζεται στην συμπεριφορά του DATA_REQUEST σήματος του STA013.

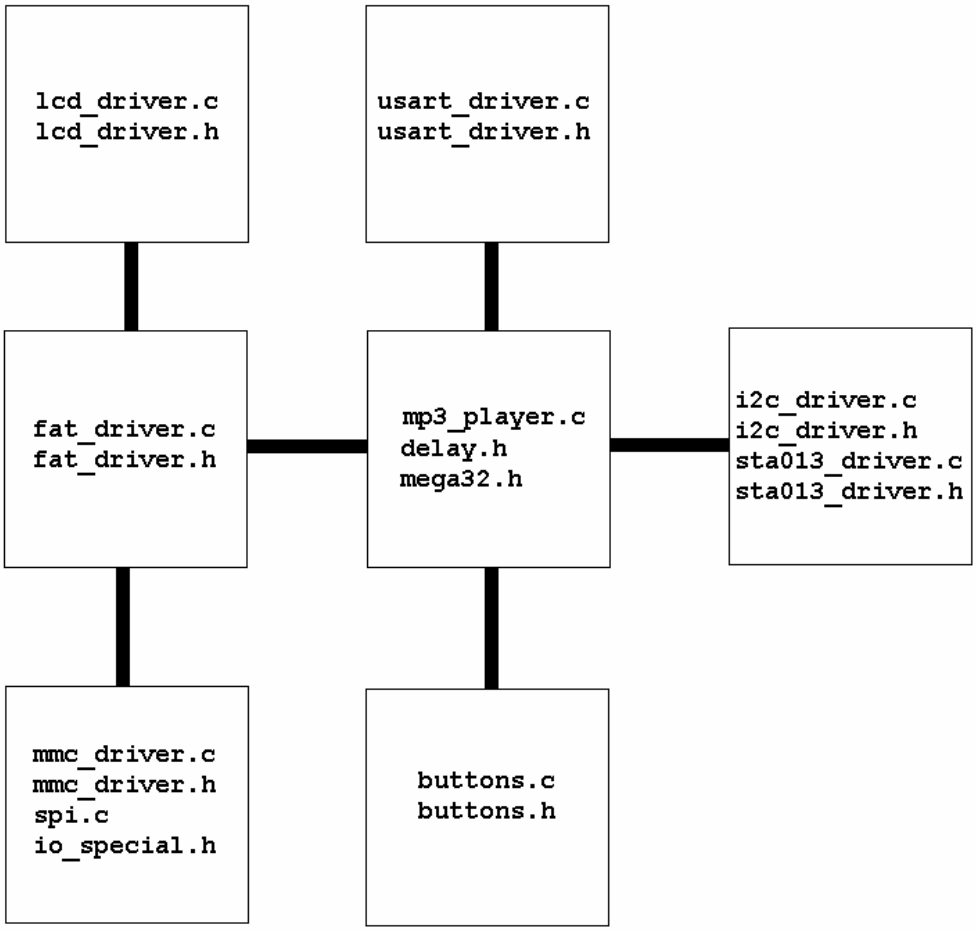


Εικόνα 6.8 : Συμπεριφορά DATA_REQUEST σήματος

Όταν το STA013 χρειάζεται περισσότερα δεδομένα υψώνει το σήμα DATA_REQUEST. Αυτό το σήμα έχει συνδεθεί έτσι ώστε να προκαλεί μία διακοπή στο κυρίως πρόγραμμα. Μέσα σε αυτή την διακοπή στέλνονται τα δεδομένα στον αποκωδικοποιητή μέχρι το chip να σταματήσει να ζητά δεδομένα. Τότε η ροή του προγράμματος θα επιστρέψει στο κυρίως βρόγχο όπου ελέγχονται οι δύο buffers (των 512 bytes) αν έχουν αδειάσει και σε αυτή την περίπτωση ξαναγεμίζουν με δεδομένα από την μνήμη.

Η οθόνη ανανεώνεται κάθε φορά που αλλάζει το τραγούδι με την καινούργια TAG πληροφορία. Επιπλέον ο έλεγχος των πλήκτρων είναι και αυτός interrupt-based αφού έχει αφιερωθεί η ρουτίνα εξυπηρέτησης μίας διακοπής κάθε 50ms να ελέγχει αν πατήθηκε κάποιο κουμπί.

Μία μορφή της διάρθρωσης του κώδικα φαίνεται παρακάτω:



Εικόνα 6.9 : Διάρθρωση προγράμματος

Παράρτημα Α

Πηγαίος Κώδικας

Α.1 – Κυρίως πρόγραμμα

mp3_player.c

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.5 Standard
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project : MP3 Player
Version : 1.0
Date    : 29/6/2005
Author  : KOSTAS RAKOPOULOS
Company : E.C.E.
Comments:
MP3 Player


Chip type           : ATmega32
Program type        : Application
Clock frequency     : 8,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 512
*****************************************************/

#include <mega32.h>
#include <delay.h>
#include <fat_driver.c>
#include <sta013_driver.c>
#include <buttons.c>
#include <usart_driver.c>

// Declare your global variables here

volatile unsigned char buffer_1[BLOCK_SIZE];
volatile unsigned char buffer_2[BLOCK_SIZE];
static bit buffer_1_full=0,buffer_2_full=0;
unsigned char check;
unsigned int mem_pointer = 0;
unsigned long int Sector_Read = 3000;

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here

    while (DATA_REQ == 1)
    {
        if ((buffer_1_full == 0) && (buffer_2_full == 0))
        {
            MMC_Read_Sector(Sector_Read,buffer_1);
```

```
            Sector_Read++;
            MMC_Read_Sector(Sector_Read,buffer_2);
            Sector_Read++;
        }

        if (mem_pointer < BLOCK_SIZE)
        {
            STA013_Send_Data(buffer_1[mem_pointer]);
            mem_pointer++;
            STA013_Send_Data(buffer_1[mem_pointer]);
            mem_pointer++;
            STA013_Send_Data(buffer_1[mem_pointer]);
            mem_pointer++;
            STA013_Send_Data(buffer_1[mem_pointer]);
            mem_pointer++;
            if (mem_pointer == BLOCK_SIZE)
            {
                buffer_1_full = 0;
            }
        }

        else
        {
            STA013_Send_Data(buffer_2[mem_pointer - BLOCK_SIZE]);
            mem_pointer++;
            STA013_Send_Data(buffer_2[mem_pointer - BLOCK_SIZE]);
            mem_pointer++;
            STA013_Send_Data(buffer_2[mem_pointer - BLOCK_SIZE]);
            mem_pointer++;
            STA013_Send_Data(buffer_2[mem_pointer - BLOCK_SIZE]);
            mem_pointer++;
            if (mem_pointer == ((2*BLOCK_SIZE) - 1))
            {
                mem_pointer = 0;
                buffer_2_full = 0;
            }
        }
    }
}

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
    Check_For_Switch();
}

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out
Func0=Out
// State7=T State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0x7F;

// Port B initialization
```

```
// Func7=Out Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In
Func0=In
// State7=0 State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0xB0;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0xFF;

// Port D initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=In Func1=In
Func0=In
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=T State1=T State0=T
PORTD=0x00;
DDRD=0xF8;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 7,813 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Rising Edge
// INT1: Off
```

```
// INT2: Off
GICR|=0x40;
MCUCR=0x03;
MCUCSR=0x00;
GIFR=0x40;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

//Initialization of LCD
LCD_Init();
LCD_Clear();
delay_ms(100);

//Initialization of USART
USART_Init();

//Initialization of MMC
SPI_Init();
PORTC=~55;
MMC_Reset();
check=0;

while (check!=1)
{
    check=MMC_Init();
    PORTC = ~check;
}

MMC_Set();
delay_ms(1000);
FAT_Read_BootRecord(buffer_1);
Sector_Read = 538;

//Load the buffers
MMC_Read_Sector(Sector_Read,buffer_1);
Sector_Read++;
buffer_1_full = 1;
MMC_Read_Sector(Sector_Read,buffer_2);
Sector_Read++;
buffer_2_full = 1;

mem_pointer = 0;


//Init of STA013
STA013_HW_Reset();
PORTC = ~STA013_Read_Reg(0x01);
STA013_Init();

do
{
    error_flag=0;
    STA013_Start();
}
```

```
while(error_flag == 1);


// Global enable interrupts
#asm ("sei")
#asm ("nop")
#asm ("nop")
#asm ("nop")
#asm ("nop")

while (1)
      {
      // Place your code here
      #asm ("cli")
      if (buffer_1_full == 0)
      {
            MMC_Read_Sector(Sector_Read,buffer_1);
            buffer_1_full = 1;
            Sector_Read++;
      }

      #asm ("sei")
      #asm ("nop")
      #asm ("nop")
      #asm ("nop")
      #asm ("nop")
      #asm ("cli")

      if (buffer_2_full == 0)
      {
            MMC_Read_Sector(Sector_Read,buffer_2);
            buffer_2_full = 1;
            Sector_Read++;
      }
      #asm("sei")

      if (Sector_Read > 567)
        Sector_Read = 528;
       };
}
```

A.2 – Ο driver για την κάρτα

mmc_driver.c

```
/*****************************************
Include file that contains the basic
routines for communication with the MMC
(Reset, Init, Set, Read, Write)
*****************************************/

#include <mmc_driver.h>
#include <io_special.h>
#include <spi.c>
#include <delay.h>

//Function for reseting the card
unsigned char MMC_Reset()
{
    unsigned char response;
    unsigned int count=0;

      MMC_Select=1;
//Raise SS

      delay_ms(250);                                          //Safety delay

      #pragma warn-
      for (count=0;count<10;count++)
      {
            SPI_Transfer_Byte(0xFF);                                //Send 10
dummy bytes 0xFF-Send 80 cycles
      }
      #pragma warn+

      count=0;

      do
      {
          response=MMC_Send_Command(GO_IDLE_STATE,0);     //Send command 0
to reset the card and put in SPI mode
          MMC_Select=1;
      //Raise SS
          if (count++ > 10)                                   //Retry for 10
times then quit
          {
                  return 0;
          }
      }
      while (response != 0x01);                               //Correct
response is 0x01

      return 1;                                               //Return success

}

//Function for initialize the card
unsigned char MMC_Init()
{
    unsigned char response;
    unsigned int count=0;
```

```
      MMC_Select=1;
//Raise SS

    delay_ms(250);                                          //Safety delay

      #pragma warn-
      for (count=0;count<10;count++)
            SPI_Transfer_Byte(0xFF);                                  //Send 10
dummy bytes 0xFF-Send 80 cycles
      #pragma warn+

      count=0;

    do
      {
          response=MMC_Send_Command(SEND_OP_COND,0);      //Send Command 1
to ensure MMC is ready
        MMC_Select=1;
      //Raise SS
        if (count++ > 500)                                  //Retry for 500
times then quit
                  return 0;
      }
      while (response);

      return 1;                                               //Return success
}

//Function for turning CRC off and setting block length
unsigned char MMC_Set()
{
    unsigned char response;

    do
      {
          response=MMC_Send_Command(CRC_ON_OFF,0);        //Send command
59 to turn CRC control off
        MMC_Select=1;
//Raise SS
      }
      while(response);

    do
    {
        response=MMC_Send_Command(SET_BLOCKLEN,BLOCK_SIZE); //Send command
16 to set Block length to 512
        MMC_Select=1;
//Raise SS
    }
    while (response);

      return 1;
      //Success
}

//Function for sending commands to the card
unsigned char MMC_Send_Command(unsigned char cmd,unsigned long int arg)
{
      unsigned char response,retry=0;

      MMC_Select=1;                                           //Raise SS
```

```
    SPI_Transfer_Byte(0xFF);                                //Send 1 dummy
bytes

      MMC_Select=0;
      SPI_Transfer_Byte(cmd | 0x40);                        //Transfer
command opcode..
      SPI_Transfer_Byte(arg>>24);                           //Transfer
argument..
      SPI_Transfer_Byte(arg>>16);
      SPI_Transfer_Byte(arg>>8);
      SPI_Transfer_Byte(arg);
      SPI_Transfer_Byte(0x95);                              //Transfer
default CRC..

      while((response = SPI_Transfer_Byte(0xFF)) == 0xFF) //Wait for right
response
      {
            if(retry++ > 8)                                   //Retry for 8
times then quit
            {
                  break;
            }
      }

      return response;                                      //Return response
}

//Function for reading a sector from the card
unsigned char MMC_Read_Sector(unsigned long int sector,unsigned char
*buffer)
{
    unsigned char response=0;
    int count;

    response = MMC_Send_Command(READ_SINGLE_BLOCK, sector<<9); //Send cmd
for read single block
    if (response!=0x00)                                     //Desired
response is 0x00
        return response;
    while (SPI_Transfer_Byte(0xFF)!=STARTBLOCK_READ)          //Wait for
data start token
        ;
    for (count=0; count<BLOCK_SIZE; count++)                  //Store in
a buffer
        *(buffer+count)=SPI_Transfer_Byte(0xFF);
    SPI_Transfer_Byte(0xFF);                                //Send
dummy bytes
    SPI_Transfer_Byte(0xFF);
    MMC_Select=1;                                           //Raise SS

    return 0;
}

//Function for writing a sector in the card
unsigned char MMC_Write_Sector(unsigned char *buffer,unsigned long int
sector)
{
    unsigned char response=0;
    int count;
```

```
    response= MMC_Send_Command(WRITE_SINGLE_BLOCK, sector<<9); //Send cmd
for write single block
    if (response!=0x00)                                          //Desired
response is 0x00
        return response;
    SPI_Transfer_Byte(0xFF);
    SPI_Transfer_Byte(STARTBLOCK_WRITE);                         //Send data
start token
    for(count=0; count<BLOCK_SIZE; count++)                     //Send data
that buffer contains
      {
            SPI_Transfer_Byte(*(buffer+count));
      }
      SPI_Transfer_Byte(0xFF);                                     //Send
dummy bytes
      SPI_Transfer_Byte(0xFF);
      response = SPI_Transfer_Byte(0xFF);                          //Get
response
      if ((response & DR_MASK) != DR_ACCEPT)                       //Check
for data correct reception
            return response;
      while(!SPI_Transfer_Byte(0xFF));
      MMC_Select=1;                                                //Raise
SS

      return 0;
}
```

mmc_driver.h

```
/*****************************************
Header file for communication with MMC
*****************************************/

//MMC Commands Opcodes
#define GO_IDLE_STATE                    0x00
#define SEND_OP_COND                     0x01
#define SEND_CSD                         0x09
#define SEND_CID                         0x0A
#define STOP_TRANSMISSION          0x0C
#define SEND_STATUS                           0x0D
#define SET_BLOCKLEN                     0x10
#define READ_SINGLE_BLOCK                0x11
#define WRITE_SINGLE_BLOCK               0x18
#define PROGRAM_CSD                           0x1B
#define SET_WRITE_PROT                   0x1C
#define CLR_WRITE_PROT                   0x1D
#define SEND_WRITE_PROT                  0x1E
#define TAG_SECTOR_START                 0x20
#define TAG_SECTOR_END                   0x21
#define UNTAG_SECTOR                     0x22
#define TAG_ERASE_GROUP_START            0x23
#define TAG_ERARE_GROUP_END              0x24
#define UNTAG_ERASE_GROUP                0x25
#define ERASE                                 0x26
#define CRC_ON_OFF                            0x3B
// R1 Response bit-defines
#define R1_BUSY                               0x80
#define R1_PARAMETER                     0x40
#define R1_ADDRESS                            0x20
#define R1_ERASE_SEQ                     0x10
```

```
#define R1_COM_CRC                              0x08
#define R1_ILLEGAL_COM                  0x04
#define R1_ERASE_RESET                  0x02
#define R1_IDLE_STATE                   0x01
// Data Start tokens
#define STARTBLOCK_READ                 0xFE
#define STARTBLOCK_WRITE                0xFE
#define STARTBLOCK_MWRITE               0xFC
// Data Stop tokens
#define STOPTRAN_WRITE                  0xFD
// Data Error Token values
#define DE_MASK                                 0x1F
#define DE_ERROR                        0x01
#define DE_CC_ERROR                             0x02
#define DE_ECC_FAIL                             0x04
#define DE_OUT_OF_RANGE                 0x04
#define DE_CARD_LOCKED                  0x04
// Data Response Token values
#define DR_MASK                                 0x1F
#define DR_ACCEPT                       0x05
#define DR_REJECT_CRC                   0x0B
#define DR_REJECT_WRITE_ERROR       0x0D

//Define block length at 512 bytes
#define BLOCK_SIZE                  0x200

unsigned char MMC_Reset();
unsigned char MMC_Init();
unsigned char MMC_Set();
unsigned char MMC_Send_Command(unsigned char,unsigned long int);
unsigned char MMC_Read_Sector(unsigned long int,unsigned char *);
unsigned char MMC_Write_Sector(unsigned char *,unsigned long int);
unsigned char SPI_Transfer_Byte(unsigned char data_out);
```

spi.c

```
/*****************************************
Include file that contains the basic
routine for communication via SPI
Initialization of SPI and Transfer Byte
*****************************************/

//Function for SPI initialization
void SPI_Init()
{
    // SPI Type: Master
    // SPI Clock Rate: 2*1000,000 kHz
    // SPI Clock Phase: Cycle Half
    // SPI Clock Polarity: Low
    // SPI Data Order: MSB First
    SPCR=0x50;
    SPSR=0x01;
}

//Function for SPI transfer byte
unsigned char SPI_Transfer_Byte(unsigned char data_out)
{
    SPDR=data_out;                  //Put data on SPDR to begin
transmission
    while (!(SPSR & (1<<SPIF)))
        ;                           //Wait for transmision end
```

```
    return SPDR;                    //Return data received
}
```

io_special.h

```
/*****************************************
Header file for special names of PORTS
*****************************************/
#define MMC_Select      PORTB.4       //Define position of MMC chip select
button
       #define SPIF            7             //Define value of SPIF
```

A.3 – Ο driver για την LCD οθόνη

lcd_driver.c

```
/*****************************************
Include file that contains the basic
routines for communication with the LCD
(Init,Read,Write,Clear,Put char,Put string)
Works for HD44780 based LCDs 2x16
*****************************************/
#include <mega32.h>
#include <lcd_driver.h>
#include <delay.h>

//Function for initializing the display..
void LCD_Init()
{
    LCD_Port_DDR=0xFF;          //Set LCD Port as output..
    delay_ms(15);
    LCD_Write(0x22,0);          //Init in 4-bit mode..
    delay_ms(5);
    LCD_Write(0x22,0);          //Init in 4-bit mode..
    LCD_Write(0x28,0);          //LCD has more than 1 lines..
    LCD_Write(0x0C,0);          //Turn the display on,the cursor off..
    LCD_Clear();                //Clear the display..
    LCD_Write(0x06,0);          //Set the cursor move direction and enable
the display..
}

//Function for writing cmd or data to the display..
void LCD_Write(unsigned char data,char cmd)
{
    LCD_Port_DDR |= 0x7F;       //Set LCD Port as output..
    LCD_RW = 0;                 //Writing enabled to lcd..

    if (cmd == 0)               //Command will be written to display..
        LCD_RS = 0;             //Set the RS low..
    else                        //Data will be written to display..
        LCD_RS = 1;             //Set the RS high..

    delay_us(1);

    //Sending the 4 MSB...
    LCD_E = 1;
    LCD_Port_Write = (LCD_Port_Write & 0xF0)+ ((data & 0xF0)>>4);
    delay_us(1);
    LCD_E = 0;

    delay_us(1);

    //Sending the 4 LSB...
    LCD_E = 1;
    LCD_Port_Write = (LCD_Port_Write & 0xF0)+ (data & 0x0F);
    delay_us(1);
    LCD_E = 0;

    //Wait until the display is not busy..
    while ((LCD_Read(0)) & 0x80);
}

//Function for reading from the lcd..
```

```
unsigned char LCD_Read(char cmd)
{
    unsigned char data;

    LCD_Port_DDR &= 0xF0;       //Set LCD data bus as input..
    LCD_RW = 1;                 //Reading enabled to lcd..

    if (cmd == 0)               //Command will be written to display..
        LCD_RS = 0;             //Set the RS low..
    else                        //Data will be written to display..
        LCD_RS = 1;             //Set the RS high..

    delay_us(1);

    //Reading the 4 MSB...
    LCD_E = 1;
    delay_us(1);
    data = (LCD_Port_Read & 0x0F)<<4;
    LCD_E = 0;

    delay_us(1);

    //Reading the 4 LSB...
    LCD_E = 1;
    delay_us(1);
    data += LCD_Port_Read & 0x0F;
    LCD_E = 0;

    return data;
}

//Function for setting cursor position
void LCD_Set_Position(unsigned char line,unsigned char pos)
{
    if (line == 0 & pos<0x10)
        LCD_Write((0x80+pos),0);
    else if (line == 1 & pos<0x10)
        LCD_Write((0xC0+pos),0);
}

//Function for clearing the display
void LCD_Clear()
{
    LCD_Write(0x01,0);          //Clear display..
    LCD_Set_Position(0,0);        //Set DD-Ram adress = 0..
}

//Function for shifting left-right the display
void LCD_Shift(unsigned char direction)
{
    if (direction == 0x01)
        LCD_Write(0x1C,0);
    else if (direction == 0x00)
        LCD_Write(0x18,0);
}

//Function for displaying a char to the display
void LCD_Print_Char(unsigned char data)
{
    LCD_Write(data,1);          //Write the char to LCD..
}
```

```
//Function for displaying 16-char long strings to the LCD
void LCD_Print_String(unsigned char *text)
{
    unsigned char count=0;

//Print the char one by one..
    do
    {
        LCD_Print_Char(*(text+count));
        count++;
    }
    while(count<0x10);
}
```

lcd_driver.h

```
/*******************************************
Header file for LCD
*******************************************/
//Definitions for connecting the LCD
#define LCD_Port_DDR    DDRA
#define LCD_Port_Write  PORTA
#define LCD_Port_Read   PINA
#define LCD_RS          PORTA.4
#define LCD_RW          PORTA.5
#define LCD_E           PORTA.6

void LCD_Init();
void LCD_Write(unsigned char data,char cmd);
unsigned char LCD_Read(char cmd);
void LCD_Clear();
void LCD_Print_Char(unsigned char data);
void LCD_Print_String(unsigned char *text);
void LCD_Set_Position(unsigned char line,unsigned char pos);
void LCD_Shift(unsigned char direction);
```

A.4 – Ο driver για το STA013

sta013_driver.c

```
#include <sta013_driver.h>
#include <i2c_driver.c>

flash unsigned char p0609[]={};

//Function gor reseting the STA013
void STA013_HW_Reset()
{
      RESET=0;
      RESET_Direction = 1;
      delay_ms(100);
      RESET_Direction = 0;
      delay_ms(500);
}

//Function for initializing the STA013
unsigned char STA013_Init()
{
    STA013_HW_Reset();
//If STA013 properly installed answer is 0xAC..
    if ((STA013_Read_Reg(0x01)) != 0xAC)
    {
        return 0;                           //Fail..
    }
    else
    {
//Do firmware configuration and tell STA013 to play..
        do
        {
            error_flag = 0;
            STA013_Update();
        }
        while (error_flag);


        return 1;                           //Success
    }
}

//Function for downloading p0609.bin to STA013
void STA013_Update()
{
    unsigned int count;
    unsigned char reg,data;

    count = 0;
    reg = p0609[0];
    data = p0609[1];

//Download the p0609 file to STA013..
//2 X 0xFF added in the end of the file..
    while ((reg != 0xFF) || (data!=0xFF))
    {
        STA013_Write_Reg(reg,data);
//Safety delay..
        if (reg == 16)
            delay_ms(1000);
```

```
        count += 2;
        delay_us(100);

        reg = p0609[count];
        data = p0609[count+1];
    }

}

//Function for writing to register of STA013
void STA013_Write_Reg(unsigned char address,unsigned char data)
{
    I2C_Start_Condition();                  //Send start condition..
    I2C_Send_Byte(0x86);                    //Send value for writing..
    I2C_Send_Byte(address);                 //Send address for writing..
    I2C_Send_Byte(data);                    //Send data..
    I2C_Stop_Condition();                   //Send stop condition..
}

//Function for reading a register from STA013..
unsigned char STA013_Read_Reg(unsigned char address)
{
    unsigned char data=0;

    I2C_Start_Condition();                  //Send start condition..
    I2C_Send_Byte(0x86);                    //Send value for writing..
    I2C_Send_Byte(address);                 //Send address for writing..
    I2C_Stop_Condition();                   //Send stop condition..
    I2C_Start_Condition();                  //Send start condition..
    I2C_Send_Byte(0x87);                    //Send value for writing..
    data = I2C_Receive_Byte();              //Read data returned..
    I2C_Stop_Condition();                   //Send stop condition..

    return data;                            //Return the value of
register..
}

//Function for STA013 start running
void STA013_Stop()
{
    STA013_Write_Reg(0x72,0x00);            //Write 0x01 to run register..
}

//Function for STA013 start running
void STA013_Run()
{
    STA013_Write_Reg(0x72,0x01);            //Write 0x01 to run register..
}

//Function for STA013 start playing
void STA013_Play()
{
    STA013_Write_Reg(0x13,0x01);            //Write 0x01 to play
register..
}

//Function for pausing STA013
void STA013_Pause()
{
   STA013_Write_Reg(0x13,0x00);
```

```
}

//Function for STA013 mute
void STA013_Mute(unsigned char on)
{
    if (on == 1)
        STA013_Write_Reg(0x14,0x01);
    else
        STA013_Write_Reg(0x14,0x00);
}

//Function for setting the volume
void STA013_Volume(unsigned char volume)
{
    STA013_Write_Reg(0x46,volume);          //Adjust volume to right
channel..
    STA013_Write_Reg(0x48,volume);          //Adjust volume to left
channel..
}

//Function for specific settings of CS4334 and starting the decoder..
void STA013_Start()
{
    STA013_Write_Reg(0x10,0x01);
    delay_ms(100);
    STA013_Write_Reg(0x54,0x03);
    STA013_Write_Reg(0x55,0x21);
    STA013_Write_Reg(0x07,0x00);
    STA013_Write_Reg(0x06,0x0c);
    STA013_Write_Reg(0x0B,0x03);
    STA013_Write_Reg(0x50,0x10);
    STA013_Write_Reg(0x51,0x00);
    STA013_Write_Reg(0x52,0x04);
    STA013_Write_Reg(0x61,0x0F);
    STA013_Write_Reg(0x64,0x55);
    STA013_Write_Reg(0x65,0x55);
    STA013_Write_Reg(0x0d,0x00);
    STA013_Write_Reg(0x0c,0x01);
    STA013_Write_Reg(0x05,0xa1);
    STA013_Write_Reg(0x18,0x04);
    STA013_Write_Reg(0x72,0x01);
    STA013_Write_Reg(0x13,0x01);
    STA013_Write_Reg(0x14,0x00);
}

//Function for sending data to STA013
void STA013_Send_Data(unsigned char data)
{
    SCKR = 1;
//Send MSB...
    if (data & 0x80)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send 7th bit...
    if (data & 0x40)
        SDI = 1;
    else
```

```
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send 6th...
    if (data & 0x20)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send 5th...
    if (data & 0x10)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send 4th...
    if (data & 0x08)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send 3th...
    if (data & 0x04)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send 2th...
    if (data & 0x02)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
    SCKR = 1;

//Send LSB...
    if (data & 0x01)
        SDI = 1;
    else
        SDI = 0;
    SCKR = 0;
}
```

sta013_driver.h

```
/*******************************************
Header file for STA013
*******************************************/
#define RESET               PORTD.3
#define RESET_Direction     DDRD.3
#define DATA_REQ            PIND.2
#define WRITE_VALUE         0x86
```

```
#define READ_VALUE          0x87
#define I2C_SDA_direction   DDRD.6
#define I2C_SDA_out         PORTD.6
#define I2C_SDA_in          PIND.6
#define I2C_SCL             PORTD.7
#define SDI                 PORTD.4
#define SCKR                PORTD.5

void STA013_HW_Reset();
unsigned char STA013_Init();
void STA013_Update();
void STA013_Write_Reg(unsigned char reg,unsigned char data);
unsigned char STA013_Read_Reg(unsigned char address);
void STA013_Stop();
void STA013_Run();
void STA013_Play();
void STA013_Pause();
void STA013_Mute(unsigned char on);
void STA013_Volume(unsigned char volume);
void STA013_Send_Data(unsigned char data);
```

i2c_driver.c

```
/*****************************************
Include file that contains the basic
routines for communication via I2C (TWI)
(Reset, Init, Set, Read, Write)
*****************************************/
#include <i2c_driver.h>

void I2C_Start_Condition()
{
// High to low transition of I2C_SDA while I2C_SCL is high
    I2C_SDA_direction = 0;
    delay_us(3);
    I2C_SCL = 1;
    delay_us(3);
    I2C_SDA_direction = 1;
    delay_us(3);
}

void I2C_Stop_Condition()
{
// Low to high transition of I2C_SDA while I2C_SCL is high
    I2C_SCL = 0;
    delay_us(3);
    I2C_SDA_direction = 1;
    delay_us(3);
    I2C_SCL = 1;
    delay_us(3);
    I2C_SDA_direction = 0;
    delay_us(3);
    I2C_SCL = 1;
    delay_us(3);
}

void I2C_Send_Byte(unsigned char data)
{
    signed char count=0;
    unsigned char error_flag;
```

```
// Clock each bit onto the SDA bus (starting with the MSB)
    for(count = 0; count < 8; count++)
    {
        I2C_SCL = 0;
        delay_us(3);
        if (data & 0x80)
        {
            I2C_SDA_direction = 0;
            delay_us(3);
        }
        else
        {
            I2C_SDA_direction = 1;
            delay_us(3);
        }
        I2C_SCL = 1;
        delay_us(3);
        data <<= 1;
    }



    // Get the ack bit
    I2C_SCL = 0;
    delay_us(3);
    I2C_SDA_direction = 0;
    delay_us(3);
    I2C_SCL = 1;
    delay_us(3);
    error_flag |= I2C_SDA_in;
    delay_us(3);

}

unsigned char I2C_Receive_Byte()
{
    signed char count=0;
    unsigned char data=0;

// Clock each bit off of the SDA bus
    I2C_SDA_out=0;

    I2C_SDA_direction = 0;
    I2C_SCL = 0;

    for(count = 7; count >= 0; count--)
    {
       delay_us(3);
       I2C_SCL = 1;
       delay_us(3);
       data = data | (I2C_SDA_in << count);           // Read the bit
while I2C_SCL is high
       I2C_SCL = 0;
    }

    I2C_SDA_direction = 1;
    delay_us(3);
    I2C_SDA_out = 0;
    delay_us(3);
    I2C_SCL = 1;
    delay_us(3);
```

```
    I2C_SDA_direction = 0;

    return data;
}
```

i2c_driver.h

```
volatile unsigned char error_flag;

void I2C_Start_Condition();
void I2C_Stop_Condition();
void I2C_Send_Byte(unsigned char data);
unsigned char I2C_Receive_Byte();
```

A.5 – O driver για τα dip-switches

buttons.c

```
/*****************************************
Include file that contains the basic
routines for the switches
*****************************************/
#include <buttons.h>

extern unsigned char buffer_1[BLOCK_SIZE];
extern unsigned char buffer_2[BLOCK_SIZE];
extern static bit buffer_1_full;
extern static bit buffer_2_full;

//Function for checking if there is a new state in buttons
void Check_For_Switch()
{
    unsigned char Switch_Status;
    static bit  Switch_Released;    //Variable for checking
    static unsigned char Volume=0;

    Switch_Status = (~PINC);

    if (Switch_Released == 0)       //If no switch was previously pussed
    {
        if (Switch_Status == 0x01)     //Case Play button pussed
        {
            Switch_Released = 1;
            STA013_Run();
            STA013_Play();
        }

        else if (Switch_Status == 0x02) //Case stop button pussed
        {
            Switch_Released = 1;
            STA013_Stop();
        }
        else if (Switch_Status == 0x04) //Case forward button pussed
        {
            Switch_Released = 1;
            Get_Next_File(buffer_1);
            Get_Next_Sector(buffer_1);
            Get_Next_Sector(buffer_2);
            buffer_1_full = 1;
            buffer_1_full = 2;
        }
        else if (Switch_Status == 0x08) //Case rewind button pussed
        {
            Switch_Released = 1;
            Get_Previous_File(buffer_1);
            Get_Next_Sector(buffer_1);
            Get_Next_Sector(buffer_2);
            buffer_1_full = 1;
            buffer_1_full = 2;
        }
        else if (Switch_Status == 0x10) //Case Mute button pussed
        {
            Switch_Released = 1;
            STA013_Mute(1);
        }
```

```
        else if (Switch_Status == 0x20) //Case Pause button pussed
        {
            Switch_Released = 1;
            STA013_Pause();
        }
        else if (Switch_Status == 0x40) //Case rewind button pussed
        {
            Switch_Released = 1;
            ((Volume > 90) ? (Volume = 90) : (Volume++));
            STA013_Volume(Volume);
        }
        else if (Switch_Status == 0x80)//Case rewind button pussed
        {
            Switch_Released = 1;
            ((Volume < 1) ? (Volume = 1) : (Volume--));
            STA013_Volume(Volume);
        }
        else                          //Case other
        {
            Switch_Released = 0;
        }
    }

    else                              //Else check if previous key released
    {
        if (Switch_Status == 0)     //If key switch released..
            Switch_Released = 0;
    }
}
```

buttons.h

```
/*****************************************
Header file for buttons
*****************************************/
#define No_Change   0x00    //Define value if no change in buttons
#define Play        0x01    //Define value if Play button pussed
#define Stop        0x02    //Define value if Stop button pussed
#define Forward     0x03    //Define value if Forward button pussed
#define Rewind      0x04    //Define value if Rewind button pussed

void Check_For_Switch();
```

A.6 – Ο driver για το FAT

fat driver.c

```
/*****************************************
Include file that contains the basic
routines for FAT16
*****************************************/
#include <mmc_driver.c>
#include <lcd_driver.c>
#include <fat_driver.h>

unsigned long int FAT_St_Add,RootDir_St_Add,Data_St_Add,Current_File_Size;
volatile unsigned int Current_Cluster=0;
volatile signed long int Current_File=0;
unsigned char Sector_Offset=0;
struct BootRecord_Info BootRecord;
unsigned char artist[0x10];
unsigned char title[0x10];

void FAT_Read_BootRecord(unsigned char *buffer)
{
    MMC_Read_Sector(0,buffer);              //Read BootRecord
    BootRecord.BytesPerSec = (unsigned int)buffer[0x0B];
    BootRecord.BytesPerSec |= (unsigned int)buffer[0x0C] << 8;
    BootRecord.SecPerClust = buffer[0x0D];
    BootRecord.ResSectors = (unsigned int)buffer[0x0E];
    BootRecord.ResSectors |= (unsigned int)buffer[0x0F] << 8;
    BootRecord.FATs = buffer[0x10];
    BootRecord.MaxRoots = (unsigned int)buffer[0x11];
    BootRecord.MaxRoots |= (unsigned int)buffer[0x12] << 8;
    BootRecord.Media = buffer[0x15];
    BootRecord.FATSecs = (unsigned int)buffer[0x16];
    BootRecord.FATSecs |= (unsigned int)buffer[0x17] << 8;
    BootRecord.SecsPerPart = (unsigned long int)buffer[0x20];
    BootRecord.SecsPerPart |= (unsigned long int)buffer[0x21] << 8;
    BootRecord.SecsPerPart |= (unsigned long int)buffer[0x22] << 16;
    BootRecord.SecsPerPart |= (unsigned long int)buffer[0x23] << 24;

    FAT_St_Add = (unsigned long int)BootRecord.ResSectors;
    RootDir_St_Add = FAT_St_Add + (unsigned long)(BootRecord.FATSecs * 2);
    Data_St_Add = RootDir_St_Add + (unsigned long)((BootRecord.MaxRoots *
32)/512);
}

unsigned long int Clust2Sec(unsigned long int Clust)
{
    return ((Clust-2) * BootRecord.SecPerClust) + Data_St_Add;
}

void Get_Next_Clust_Add(unsigned char *buffer)
{
    unsigned long int Sector_Read;
    unsigned int Cluster,Offset;

    Cluster = Current_Cluster;
    Sector_Read = (Cluster / 256) + FAT_St_Add;
    MMC_Read_Sector(Sector_Read, buffer);
    Offset = (Cluster & 255) * 2;
    Cluster  = ((unsigned int)buffer[Offset]);
    Cluster |= ((unsigned int)(buffer[Offset+1]) << 8);
```

```
    Current_Cluster = Cluster;
}

void Get_Next_Sector(unsigned char *buffer)
{
    if (Sector_Offset > BootRecord.SecPerClust)
    {
        Sector_Offset = 0;
        Get_Next_Clust_Add(buffer);
        if (Current_Cluster >= 0xFFF8)
            Get_Next_File(buffer);
    }

    else
    {
        MMC_Read_Sector(Clust2Sec(Current_Cluster)+Sector_Offset,buffer);
        Sector_Offset++;
    }
}

void Get_Next_File(unsigned char *buffer)
{
    signed long int File;
    unsigned long int Sector_Read;

    File = Current_File + 1;

    while (1)
    {
        if ((File * 0x20) >= BootRecord.MaxRoots)
            File = 0;

        Sector_Read = (File / 0x10) + RootDir_St_Add;
        MMC_Read_Sector(Sector_Read, buffer);

        if ((buffer[(File % 0x10) * 0x20] == 0xE5) || (buffer[(File %
0x10)* 0x20] == 0x00))
        {
            File++;
        }

        else
        {
            if (buffer[((File % 0x10) * 0x20) + 0x0B] == 0x0F)
            {
                File++;
            }

            else
            {
                Current_Cluster  = (unsigned int)buffer[((File % 0x10) *
0x20) + 0x1A];
                Current_Cluster |= ((unsigned int)buffer[((File % 0x10) *
0x20) + 0x1B] << 8);
                Current_File_Size  = (unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1C];
                Current_File_Size |= ((unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1D] << 8);
                Current_File_Size |= ((unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1E] << 16);
```

```
                Current_File_Size |= ((unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1F] << 24);
                Current_File = File;
                break;
            }
        }
    }
}

void Get_Previous_File(unsigned char *buffer)
{
    unsigned int File;
    unsigned long int Sector_Read;

    ((Current_File == 0) ? (File = Current_File) : (File = Current_File -
1));

    while (1)
    {
        Sector_Read = (File / 0x10) + RootDir_St_Add;
        MMC_Read_Sector(Sector_Read, buffer);

        if (buffer[(File % 0x10)*0x20] == 0xE5 || buffer[(File %
0x10)*0x20] == 0x00)
        {
            File--;
        }

        else
        {
            if (buffer[((File % 0x10) * 0x20) + 0x0B] == 0x0F)
            {
                File--;
            }

            else
            {
                Current_Cluster  = (unsigned int)buffer[((File % 0x10) *
0x20) + 0x1A];
                Current_Cluster |= (unsigned int)buffer[((File % 0x10) *
0x20) + 0x1B] << 8;
                Current_File_Size  = (unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1C];
                Current_File_Size |= (unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1D] << 8;
                Current_File_Size |= (unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1E] << 16;
                Current_File_Size |= (unsigned long int)buffer[((File %
0x10) * 0x20) + 0x1F] << 24;
                Current_File = File;
                break;
            }
        }
    }
}

void Show_Tag_Info(unsigned char *buffer)
{
    unsigned char Read = 1, Count1 = 0, i=0, check=0;
    unsigned int Prev_Cluster,Next_Cluster,Offset_New,Count2;
    unsigned long int Sector_Read;
```

```
    Prev_Cluster = Current_Cluster;


    do
    {
        Sector_Read = (Prev_Cluster / 256) + FAT_St_Add;

        if (Read == 1)
        {
            MMC_Read_Sector(Sector_Read, buffer);
        }

        Offset_New = (Prev_Cluster & 255) * 2;
        Next_Cluster  = ((unsigned int)buffer[Offset_New]);
        Next_Cluster |= ((unsigned int)(buffer[Offset_New+1]) << 8);

        if ((Next_Cluster / 256) == (Prev_Cluster / 256))
        {
            Read = 0;
        }

        else
        {
            Read = 1;
        }

        if (Next_Cluster <= 0xFFF7)
        {
            Prev_Cluster = Next_Cluster;
        }
    }
    while (Next_Cluster <= 0xFFF7);

    do
    {
        MMC_Read_Sector(Clust2Sec(Prev_Cluster)+Count1,buffer);
        for (Count2 = 0; Count2 < 0x200; Count2++)
        {
            if ((buffer[Count2] == 'T') && (buffer[Count2+1] == 'A') &&
(buffer[Count2+2] == 'G'))
            {
                while (i < 0x10)
                {
                    title[i] = buffer[Count2+3+i];
                    artist[i] = buffer[Count2+33+i];
                    i++;
                }
                check = 1;
            }
        }

        Count1++;

        if (Count1 > BootRecord.SecPerClust)
        {
            Count1 = 0;
        }
    }
    while (check != 1);
```

```
    LCD_Set_Position(0,0);
    LCD_Print_String(artist);
    LCD_Set_Position(1,0);
    LCD_Print_String(title);
}
```

fat driver.h

```
/*****************************************
Header file for FAT16
*****************************************/
struct BootRecord_Info
{
    unsigned int            BytesPerSec;  // bytes per sector
    unsigned char           SecPerClust;  // sectors per cluster
      unsigned int          ResSectors;         // number of reserved
sectors
      unsigned char         FATs;               // number of FATs
      unsigned int        MaxRoots;       //Maximum Root dir entries
      unsigned char       Media;          //Media descriptor
      unsigned int          FATSecs;            // number of sectors per
FAT
      unsigned long int SecsPerPart;      // sectors per track
};

void FAT_Read_BootRecord(unsigned char *buffer);
unsigned long int Clust2Sec(unsigned long int Clust);
void Get_Next_Clust_Add(unsigned char *buffer);
void Get_Next_Sector(unsigned char *buffer);
void Get_Next_File(unsigned char *buffer);
void Get_Previous_File(unsigned char *buffer);
void Show_Tag_Info(unsigned char *buffer);
```

A.7 – Ο driver για την USART

usart driver.c

```
/*****************************************
Include file that contains the basic
routines for the USART communication
*****************************************/
#include <usart_driver.h>

//Function for initializing the USART
void USART_Init()
{
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: Off
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x08;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0xCF;
}

//Function for writing chars to USART
void USART_Put_Char(unsigned char data)
{
//Wait for empty transmit buffer..
    while (!(UCSRA & 0x20))
        ;

//Put data into buffer, sends the data..
    UDR = data;
}

//Function for reading chars from USART
unsigned char USART_Get_Char()
{
//Wait for data to be received..
    while (!(UCSRA & 0x80))
        ;

//Return data received..
    return UDR;
}
```

usart driver.c

```
//Header File for USART transmission..
void USART_Init();
void USART_Put_Char(unsigned char data);
unsigned char USART_Get_Char();
```

Παράρτημα Β

Περιγραφή του CodeVision AVR

Β.1 – Εισαγωγή

Σκοπός αυτού του παραρτήματος είναι να γίνει μία σύντομη περιγραφή του προγράμματος CodeVision AVR C Compiler μέσω ενός απλού παραδείγματος, έτσι ώστε να μπορεί ο αναγνώστης μετά την ανάγνωση αυτή να χειρίζεται το πρόγραμμα και να προγραμματίζει μικροελεγκτές AVR στην γλώσσα προγραμματισμού C.

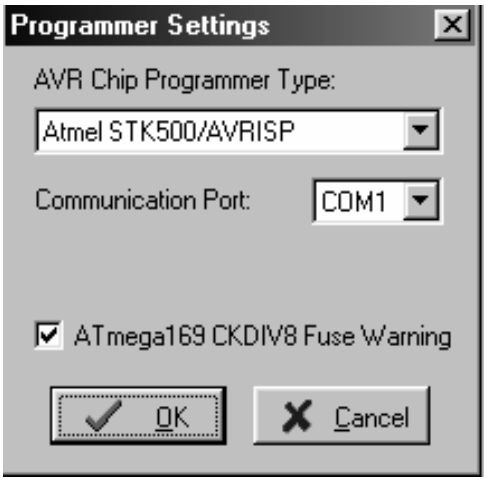
Το CodeVision AVR είναι ένας από τους πολλούς C compilers που υπάρχουν στην αγορά διαθέσιμοι για τους μικροελεγκτές AVR. Αυτό που το κάνει να ξεχωρίζει είναι το εκπληκτικό γραφικό περιβάλλον προγραμματισμού και η ευκολία στην χρήση του. Αντίθετα με πολλούς compilers της C που έχουν αρχικά κατασκευαστεί για άλλους σκοπούς και τροποποιήθηκαν για να συμπεριλάβουν το σετ εντολών του μικροελεγκτή AVR, το CodeVision AVR έχει γραφεί ειδικά για τον μικροελεγκτή με αποτέλεσμα να εκμεταλλεύεται όλες τις δυνατότητες του. Σαν αποτέλεσμα παράγει ιδανικό πηγαίο κώδικα χωρίς σπατάλες. Τέλος έχει τον CodeWizard AVR generator, ένα πολύτιμο εργαλείο για την γρήγορη δημιουργία καινούργιων projects.

Β.2 – Δημιουργία ενός απλού project στο CodeVision AVR

Παρακάτω θα περιγράψουμε την δημιουργία ενός project με στόχο ένα ATmega32 (και το STK500) το οποίο θα επιδεικνύει μία απλή λειτουργία. Αυτό θα είναι ένα πρόγραμμα το οποίο θα αναβοσβήνει ένα led του STK500 κάθε 500 ms και θα ανάβει ένα άλλο όταν συμβαίνει μία εξωτερική διακοπή.

Β.2.1 – Ρύθμιση του CodeVision AVR για την λειτουργία του με το STK500

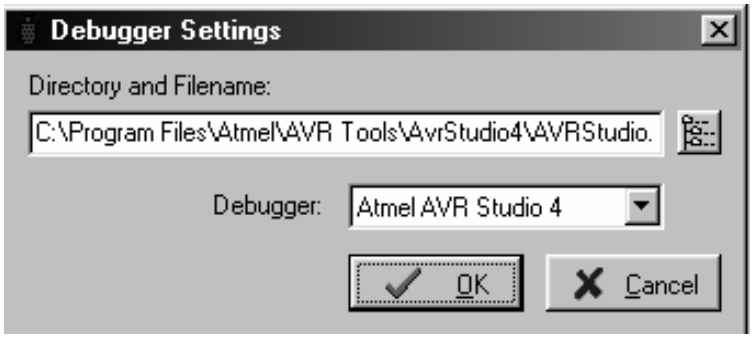
Αφού το πρόγραμμα έχει εγκατασταθεί στο default directory (C:\cvavr) πρέπει να ρυθμιστούν κάποιες σημαντικές παράμετροι. Αυτό γίνεται με τα εξής βήματα. Αρχικά εκτελείται το πρόγραμμα και επιλέγεται η επιλογή **Settings | Programmer.** Το παρακάτω παράθυρο πρόκειται να ανοίξει.



Εικόνα Β.1 : Programmer Settings

Εδώ πρέπει να βεβαιωθεί ο χρήστης ότι το STK500 είναι ο στόχος και ότι χρησιμοποιείται η σωστή σειριακή θύρα από το πρόγραμμα.

Έπειτα πρέπει να επιλεχτεί ο debugger που πρόκειται να χρησιμοποιηθεί. Επιλέγεται **Settings | Debugger** και ανοίγει το επόμενο παράθυρο.

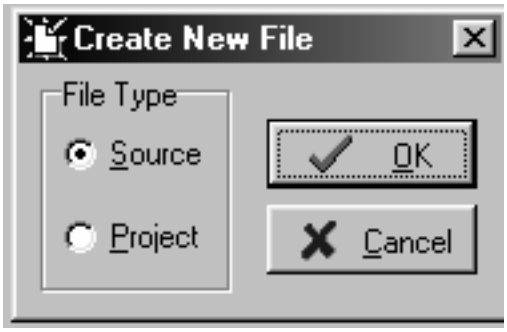


Εικόνα Β.2 : Debugger Settings

Εδώ επιλέγεται το AVR Studio 4 της ATMEL.

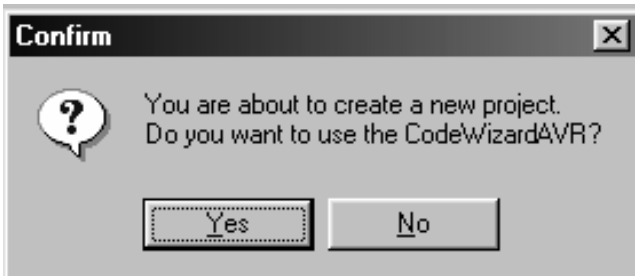
Β.2.2 – Δημιουργία ενός νέου project

Για να δημιουργηθεί ένα καινούργιο project, επιλέγεται **File | New.** Το ακόλουθο παράθυρο θα εμφανιστεί.



Εικόνα Β.3 : Δημιουργία καινούργιου project

Επιλέγεται project και εμφανίζεται το επόμενο παράθυρο.

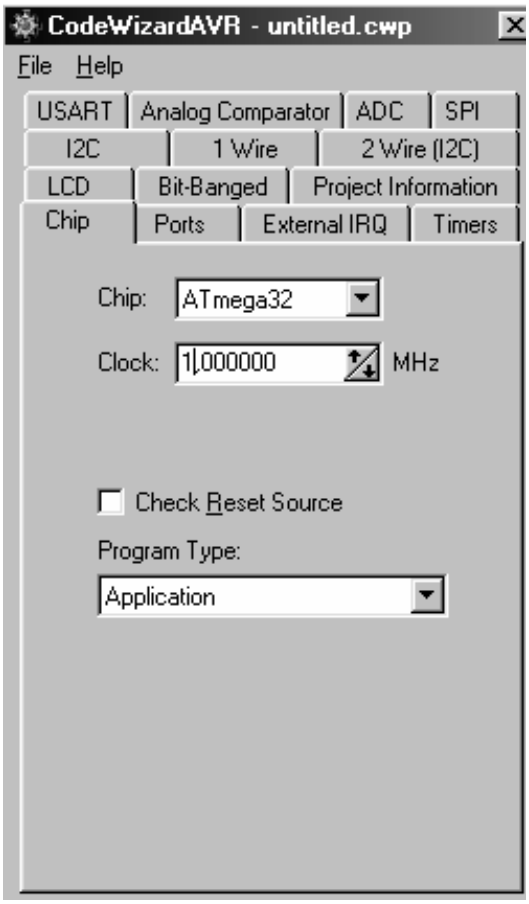


Εικόνα Β.4 : Επιλογή χρησιμοποίησης CodeWizard

Επιλέγεται yes για την χρησιμοποίηση του CodeWizard generator.

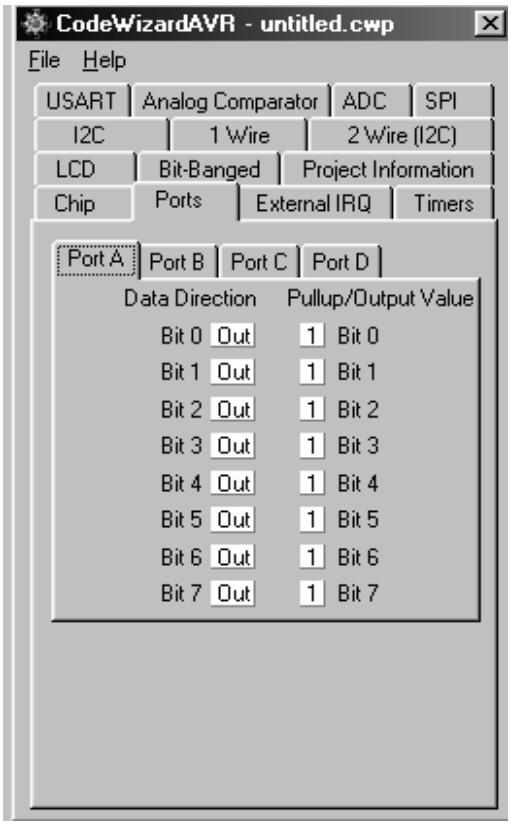
Β.2.3 – Χρησιμοποίηση του CodeWizard generator

Ο CodeWizard generator απλοποιεί το γράψιμο αρχικοποίησης για κάθε μικροελεγκτή AVR. Το επόμενο παράθυρο ανοίγει και επιλέγεται ο AVR ATmega32 και ρολόι το εσωτερικό του στα 1 MHz.



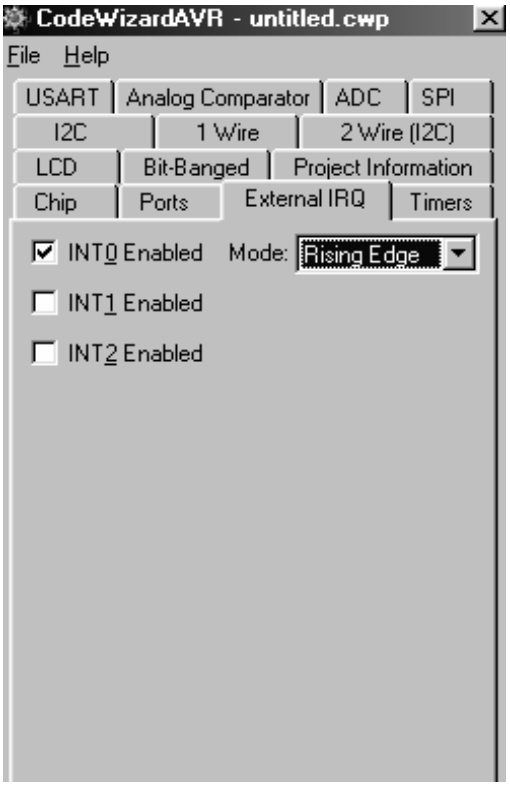
Εικόνα Β.4 : Ο CodeWizard generator

Έπειτα επιλέγεται το tab PORTA το οποίο και έχουμε συνδέσει με τα leds του αναπτυξιακού. Εδώ κάνουμε τα pin της πόρτας από είσοδο-έξοδο.



Εικόνα Β.5 : Επιλογή διεύθυνσης PORTA

Για το project θα χρησιμοποιήσουμε και την εξωτερική διακοπή 0.

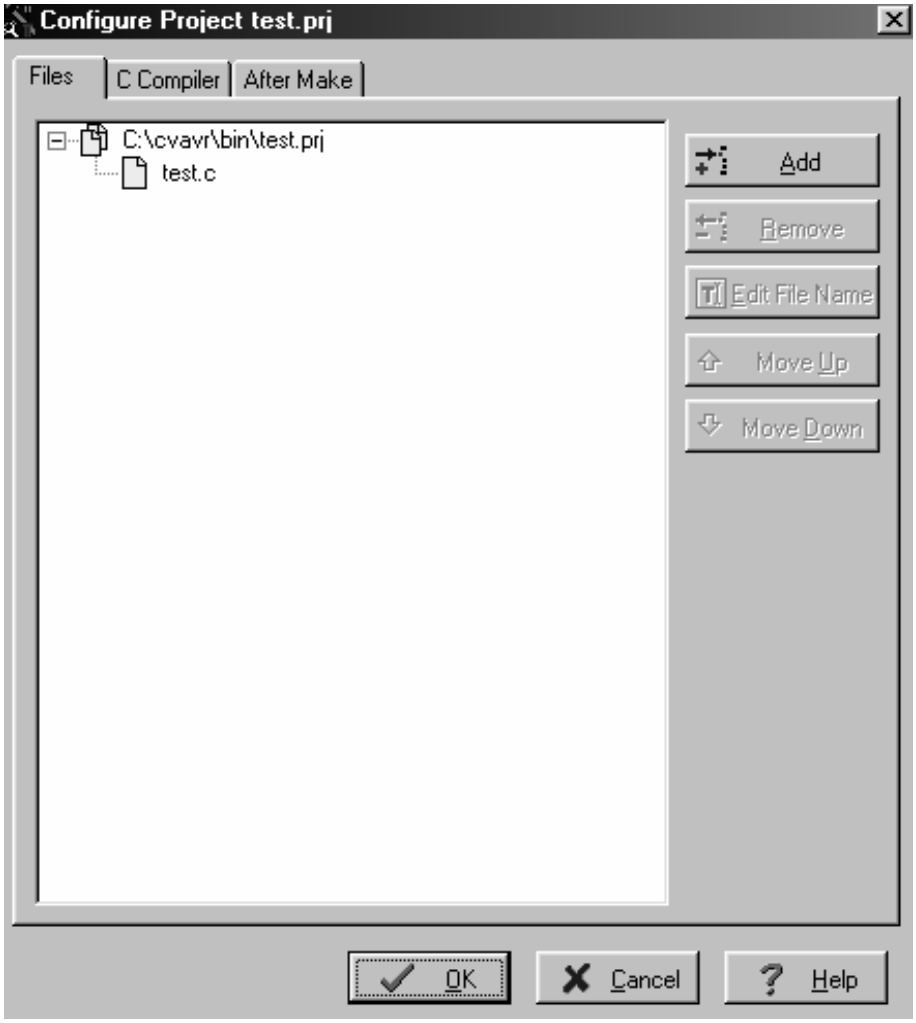


Εικόνα Β.6 : Ενεργοποίηση εξωτερικής διακοπής 0

Έπειτα από αυτές τις ρυθμίσεις επιλέγεται **File | Generate, Save and Exit** και ο CodeWizard θα δημιουργήσει τον σκελετό του προγράμματος. Σώζεται το project και εμφανίζεται το περιβάλλον προγραμματισμού. Ο κώδικας στο τέλος του παραρτήματος έχει την προσθήκη μέσα στην εξυπηρέτηση της διακοπής να αλλάζει το state ενός led και να αναβοσβήνει ένα άλλο led στο κυρίως πρόγραμμα.

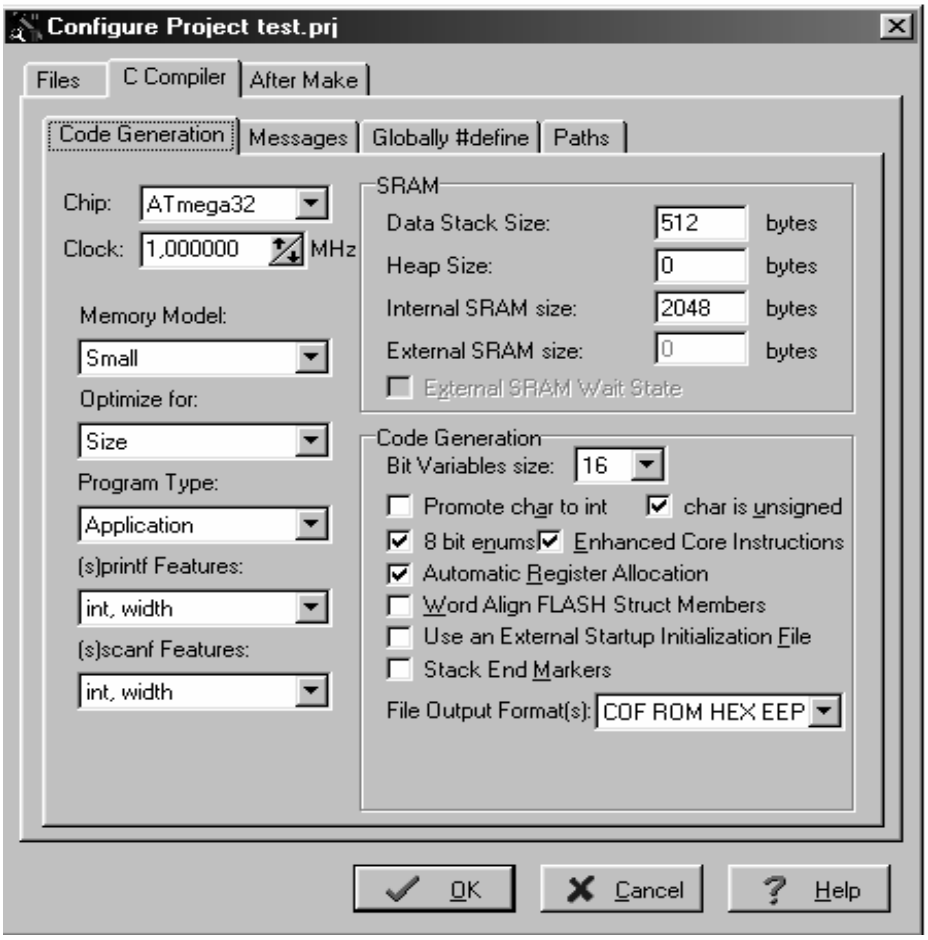
Β.2.4– Ρυθμίσεις του project

Επιλέγεται **Project | Configure.** Το επόμενο παράθυρο εμφανίζεται:



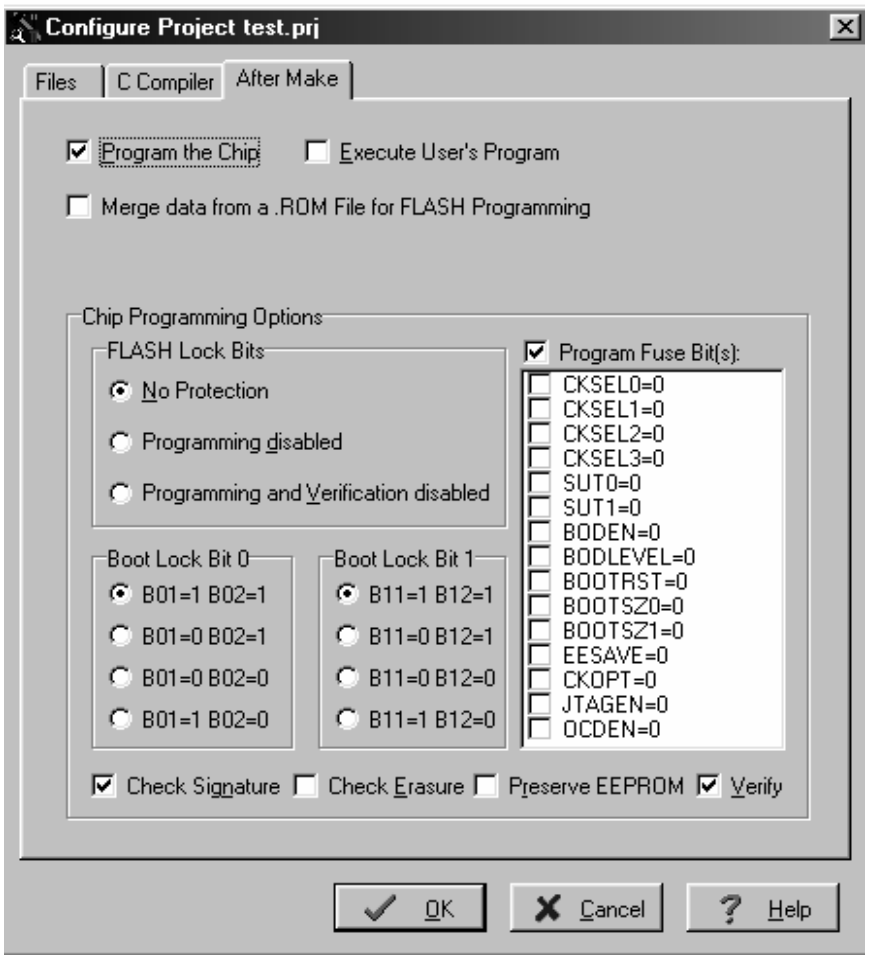
Εικόνα Β.7 : Project configuration

Εδώ μπορούν να προστεθούν και να αφαιρεθούν αρχεία στο project με τα κουμπιά **Add** και **Remove.** Έπειτα επιλέγεται το tab **C Compiler.**



Εικόνα Β.8 : Επιλογές C Compiler

Εδώ μπορούν να αλλάξουν όλες οι επιλογές του project. Έπειτα επιλέγεται το tab **After Make.** Το επόμενο παράθυρο εμφανίζεται.

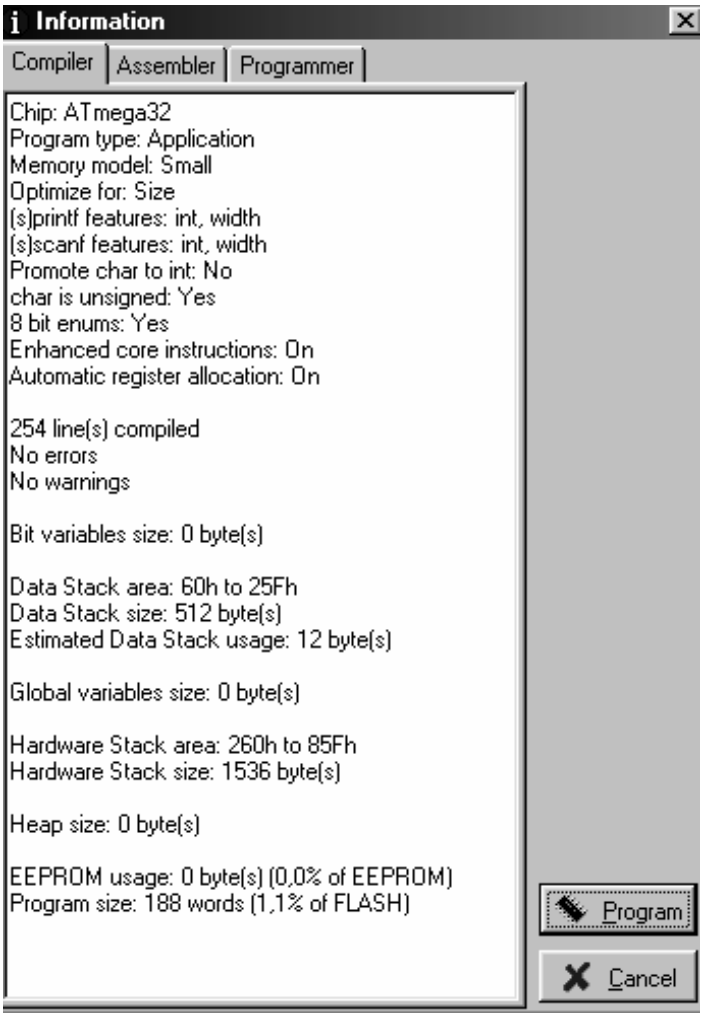


Εικόνα Β.9 : After make tab

Εδώ επιλέγεται το **Program the chip** έτσι ώστε μετά το compilation να προγραμματίζεται το chip αμέσως.

Β.2.5 – Making the project

Έπειτα από όλα αυτά επιλέγεται το **Project | Make.** Αυτή η επιλογή δημιουργεί τον κώδικα για τον προγραμματισμό του chip. Το επόμενο παράθυρο εμφανίζεται.



Εικόνα Β.10 : Project compiled

Το STK500 προγραμματίζεται με το πλήκτρο **Program.** Έπειτα από αυτό το πρόγραμμα αρχίζει να τρέχει αυτόματα στο αναπτυξιακό. Το πρόγραμμα επίδειξης ακολουθεί ακριβώς παρακάτω.

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.5 Standard
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project : Demonstration Program
Version : 0.1
Date    : 6/7/2005
Author  : Kostas Rakopoulos
Company : E.C.E.
Comments:
Demonstration Program of CodeVision AVR


Chip type           : ATmega32
Program type        : Application
Clock frequency     : 1,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 512
*****************************************************/

#include <mega32.h>
#include <delay.h>

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here
   PORTA.1 = PORTA.1 ^ 1;
}

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out
Func0=Out
// State7=1 State6=1 State5=1 State4=1 State3=1 State2=1 State1=1 State0=1
PORTA=0xFF;
DDRA=0xFF;

// Port B initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
```

```
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Rising Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x03;
MCUCSR=0x00;
GIFR=0x40;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")

while (1)
    {
    // Place your code here
    PORTA.0 = PORTA.0 ^ 1;    //Toggle led
    delay_ms(500);          //Wait 500 ms
    };
}
```

Βιβλιογραφία

[1].- K. Z. Πεκμεστζή – Συστήματα Μικροϋπολογιστών

[2].- Barnett, Cox & O' Cull – Embedded C programming and the ATMEL AVR

[3].- Brian W. Kernighan, Dennis M. Ritchie – Η γλώσσα προγραμματισμού C

[4].- SanDisk MultiMediaCard and Reduced-Size MultiMediaCard - Product Manual Version 1.0

[5].- www.atmel.com

[6].- www.avrfreaks.net

[7].- LCD display unit user's manual – SHARP

[8].- www.pjrc.com