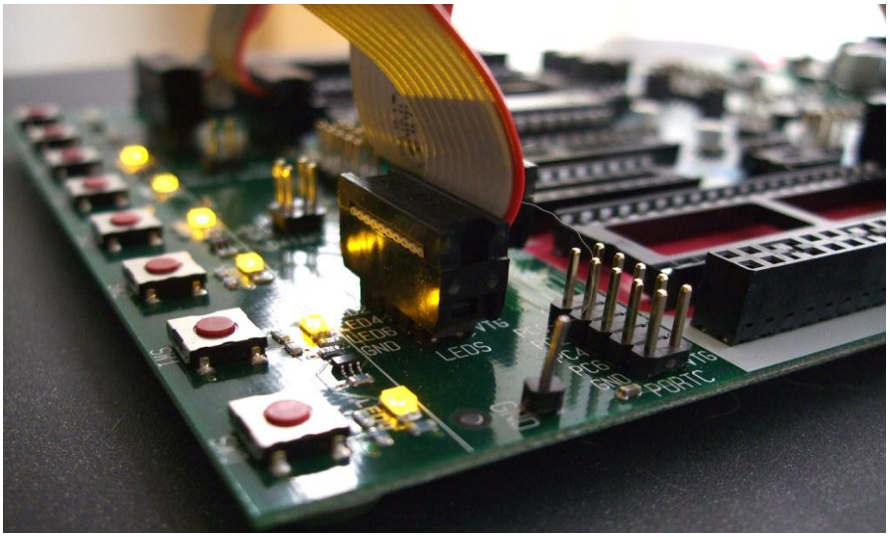
ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ ΜΕ ΤΗΝ ΣΕΙΡΑ ΜΙΚΡΟΕΛΕΓΚΤΩΝ MEGA ΤΗΣ ATMEL



ΣΠΟΥΔΑΣΤΗΣ : **ΚΑΛΑΜΑΡΑΣ ΑΛΕΞΑΝΔΡΟΣ**

ΕΙΣΗΓΗΤΗΣ ΚΑΘΗΓΗΤΗΣ : **ΣΑΠΟΥΝΙΔΗΣ ΘΕΟΔΟΣΙΟΣ**

ΘΕΣΣΑΛΟΝΙΚΗ

2009

ΠΕΡΙΕΧΟΜΕΝΑ

ΣΕΛΙΔΑ

ΠΕΡΙΛΗΨΗ……...……1

ΕΙΣΑΓΩΓΗ…….…….…2

ΟΙ ΕΝΝΟΙΕΣ ΤΟΥ ΜΙΚΡΟΕΛΕΓΚΤΗ ,ΤΟΥ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΗ ΚΑΙ ΤΟΥ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΗ…….…….....3

ΟΙ ΜΙΚΡΟΕΛΕΓΚΤΕΣ AVR ΤΗΣ ATMEL…….……..…..4

ΤΑ ΣΥΣΤΑΤΙΚΑ ΜΕΡΗ ΕΝΟΣ ΜΙΚΡΟΕΛΕΓΚΤΗ…….…...6

Η ΜΝΗΜΗ……..……..…11

ΚΑΤΑΧΩΡΗΤΕΣ…….…….....……..12

ΔΙΑΚΟΠΕΣ (INTERRUPTS)……...……....15

ΔΙΑΚΟΠΗ ΕΠΑΝΝΕΚΙΝΗΣΗΣ (RESET)……...…...19

WATCHDOG TIMER……...….....20

ΑΝΑΛΟΓΙΚΕΣ ΔΙΑΣΥΝΔΕΣΕΙΣ…….…...……23

Η ΜΟΝΑΔΑ ΤΟΥ ΜΕΤΑΤΡΟΠΕΑ ΤΑΣΗΣ (ADC PERIPHERAL)……..…..……24

ΧΡΟΝΙΣΤΕΣ - ΜΕΤΡΗΤΕΣ (TIMER/COUNTERS)…….....….....27

ΟΙ ΠΡΟΔΙΑΙΡΕΤΕΣ ΤΩΝ ΧΡΟΝΙΣΤΩΝ…….……28

Ο TIMER 0…….....29

Ο TIMER 1…….....31

Ο ΠΡΟΔΙΑΙΡΕΤΗΣ ΤΟΥ TIMER1……..31

Η ΛΕΙΤΟΥΡΓΙΑ ΣΥΛΛΗΨΗΣ ΤΟΥ TIMER1(TIMER1 CAPTURE)……..32

ΛΕΙΤΟΥΡΓΙΑ ΕΞΟΔΟΥ ΣΥΓΚΡΙΣΗΣ ΤΟΥ TIMER1 (OUTPUT COMPARE)… 35

ΔΙΑΜΟΡΦΩΣΗ ΠΛΑΤΟΥΣ ΠΑΛΜΩΝ ΜΕ ΤΟΝ TIMER1 (PWM)……...…….37

ΑΠΕΙΚΟΝΙΣΗ ΣΕ ΕΝΔΕΙΚΤΗ LED 7 ΤΟΜΕΩΝ……....…….41

ΟΔΗΓΗΣΗ ΚΙΝΗΤΗΡΩΝ ΜΕ ΤΟΥΣ ΜΙΚΡΟΕΛΕΓΚΤΕΣ AVR……..….....43

ΟΔΗΓΗΣΗ ΚΙΝΗΤΗΡΩΝ DC……..….…..…43

ΟΔΗΓΗΣΗ ΒΗΜΑΤΙΚΟΥ ΚΙΝΗΤΗΡΑ…….…..…..44

ΟΔΗΓΗΣΗ ΚΙΝΗΤΗΡΑ SERVO……...……..…46

ΟΔΗΓΗΣΗ ΟΘΟΝΗΣ ΥΓΡΩΝ ΚΡΥΣΤΑΛΩΝ……...48

ΣΕΙΡΙΑΚΗ ΕΠΙΚΟΙΝΩΝΙΑ ΜΕ ΤΟ USART……...….....50

ΤΟ ΠΡΩΤΟΚΟΛΛΟ SPI……..…55

ΤΟ ΠΡΩΤΟΚΟΛΛΟ I^2C……..60

Η ΔΟΜΗ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ I^2C……61

ΤΟ ΠΡΩΤΟΚΟΛΛΟ 1 WIRE……..…65

ΑΙΣΘΗΤΗΡΑΣ ΘΕΡΜΟΚΡΑΣΙΑΣ 1 WIRE (DS18B20)……..……..…….....66

Ο ΜΕΤΑΓΛΩΤΤΙΣΣΤΗΣ CODEVISION ……..……..…...……70

Ο ΠΡΟΓΡΑΜΜΑΤΙΣΤΗΣ PonyProg2000……..……..….……..…75

Η ΠΛΑΚΕΤΑ ΕΦΑΡΜΟΓΩΝ ΓΙΑ ΤΟΥΣ AVR…….....……..…….....76

ΒΙΒΛΙΟΓΡΑΦΙΑ…….....…80

ΠΑΡΑΡΤΗΜΑ……...…..81

ATMEGA16…….…….....81

L293D…….……...85

ΠΕΡΙΛΗΨΗ

Στην ακόλουθη διπλωματική εργασία έχουν αναπτυχθεί εκπαιδευτικές εφαρμογές, οι οποίες θα βοηθήσουν τον αρχάριο χρήστη των μικροελεγκτών AVR να εξοικειωθεί μαζί τους προγραμματίζοντας με την βοήθεια της γλώσσας προγραμματισμού C. Στις επόμενες σελίδες γίνεται λόγος για την εσωτερική δομή των μικροελεγκτών ,καθώς και για τις περιφερειακές μονάδες του μικροελεγκτή (timers ,ADC ,PWM) όπου δίνονται παραδείγματα εκτελέσιμου κώδικα μαζί με την κυκλωματική διασύνδεση των εφαρμογών. Στην πλακέτα των εφαρμογών που κατασκέυασα στα πλαίσια της εργασίας, έχουμε την ευκαιρία να μελετήσουμε την διασύνδεση ενός ATMEGA16 με διάφορες μονάδες όπως με οθόνη LCD, led displays ,με το L293, σειριακή επικοινωνία με το USART ,ρολόι πραγματικού χρόνου καθώς και εφαρμογές σε πρωτόκολλα επικοινωνίας όπως το SPI, I^2C και 1 wire.

INTRODUCTION

In the following diplomatic project, the educational applications that have been developed , will help the novice user to familiarize with the AVR microcontrollers, programming it, with the help of C programming language. In the following pages there is a reference about the internal structure of microcontrollers and the peripheral units of microcontroller (timers, ADC, PWM) will be analyzed with examples of executable code with the applications schematic connection. In the circuit board developed for the project, the applications that are reported give us the opportunity to study the connection of ATMEGA16 with various external units such as screen LCD, led displays, the L293, serial communication with the USART, real time clock and applications based in communication protocols such as the SPI, I2C and 1 wire.

ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια η Atmel ήταν πρωτοπόρος στην ανάπτυξη μνήμης τεχνολογίας flash. Οι μνήμες flash είναι μη πτητικές, επαναπρογραμματιζόμενες που συχνά χρησιμοποιούνται σε διάφορα προιόντα τεχνολογίας όπως ψηφιακές φωτογραφικές μηχανές, φορητές συσκευές αναπαραγωγής ήχου και μητρικές κάρτες υπολογιστών. Αυτήν την τεχνολογία μνήμης ενσωμάτωσε στους μικροελεγκτές της τους AVR. Οι μικροελεγκτές ολοένα χρησιμοποιούνται περισσότερο στην καθημερινή μας ζωή. Εκτιμάται πως ως το τέλος της τρέχουσας δεκαετίας ένα μέσο σπίτι θα περιλαμβάνει ουσιαστικά συστήματα των 50 έως και 100 συνολικά μικροελεγκτών, οι οποίοι θα ελέγχουν διάφορες συσκευές όπως είναι τα ψηφιακά τηλέφωνα,οι φούρνοι μικροκυμάτων, οι συσκευές αναπαραγωγής εικόνας, οι συσκευές τηλεοράσεως καθώς επισης και τα αντίστοιχα τηλεχειριστήρια, τα πληντύρια πιάτων, τα συστήματα ασφαλείας και διάφορα άλλα. Είναι γεγονός πως οι μικροελεγκτές είναι πιο προσιτοί στην χρήση τους σε συστήματα ελέγχου σε σχέση με ειδικά ψηφιακά κυκλώματα που υλοποιούν πολύπλοκες μηχανές καταστάσεων και πολύ πιο οικονομικοί σε περιπτώσεις αναβάθμισης. Οι ιδιότητες αυτές που απορρέουν από την χρήση των μικροελεγκτών οδηγούν σε γενικά οικονομικότερες λύσεις και μεγαλύτερα κέρδη για αυτόν που θα τους επιλέξει για την ανάπτυξη εφαρμογών. Εκεί άλλωστε οφείλεται κυρίως και η ραγδαία ανάπτυξη των μικροελεγκτών. Παρατηρείται γενικότερα μια μαζική χρήση των μικροελεγκτών σε επίπεδο ειδικών διατάξεων συστημάτων και διαφόρων εξαρτημάτων που διατίθονται στο εμπόριο, γεγονός το οποίο φανερώνει μια σημαντική πηγή μεγάλου οικονομικού οφέλους. Συνεπώς κρίνεται απαραίτητη η γνώση των μικροελεγκτών για την ανάπτυξη αποδοτικών ανταγωνιστικών εφαρμογών.

ΟΙ ΕΝΟΙΕΣ ΤΟΥ ΜΙΚΡΟΕΛΕΓΚΤΗ ,ΤΟΥ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΗ ΚΑΙ ΤΟΥ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΗ

Έχουμε συνηθίσει να ακούμε τους όρους αυτούς χωρίς κανέναν ιδιαίτερο διαχωρισμό. Ωστόσο, κάθε ένας από τους παραπάνω όρους έχει διακριτή σημασία και είναι σημαντικό να κατανοήσουμε τις μεταξύ τους διαφορές στο σημείο αυτό.

Ένας μικροεπεξεργαστής είναι ουσιαστικά μια μονάδα κεντρικής επεξεργασίας (ή CPU) υλοποιημένη μέσα σε ένα ολοκληρωμένο κύκλωμα. Πριν από πολλά χρόνια, μια τέτοια μονάδα επεξεργασίας - CPU - σχεδιάζονταν με τη βοήθεια πολλών διακριτών ολοκληρωμένων κυκλωμάτων μεσαίας και μεγάλης κλίμακας ολοκλήρωσης (MSI και LSI). Η γνωστή εταιρία Intel, παρουσιάζοντας το 4004, έθεσε όλες τις απαραίτητες μονάδες που περιλαμβάνει τυπικά μια CPU (όπως την αριθμητική λογική μονάδα ή ALU, τον αποκωδικοποιητή) εντολών, τους καταχωρητές, κυκλώματα ελέγχου διαύλων, κ.λ.π.), εντός ενός απλού ολοκληρωμένου κυκλώματος (chip) και έτσι "γεννήθηκε" ο μικροεπεξεργαστής. Ο 4004 ήταν ένας απλός μικροεπεξεργαστής των 4 δυαδικών ψηφίων (bits) - (δηλαδή είχε την δυνατότητα επεξεργασίας δεδομένων υπό την μορφή λέξεων των 4-bits τη φορά), ο οποίος σχεδιάστηκε για να χρησιμοποιηθεί ως μονάδα επεξεργασίας αριθμών σε μια αριθμομηχανή.

Όταν μια τέτοια διάταξη, όπως ένας μικροεπεξεργαστής, μαζί με τα απαραίτητα κυκλώματα υποστήριξης-περιφερειακές μονάδες εισόδου - εξόδου και μνήμη προγράμματος και δεδομένων - τοποθετηθούν μαζί στο ίδιο υλικό, έτσι ώστε να σχηματίσουν έναν στοιχειώδη υπολογιστή, ειδικότερα σε εφαρμογές ελέγχου και λήψης δεδομένων, η διάταξη που προκύπτει καλείται μικροϋπολογιστής.

Με άλλα λόγια αν θέλαμε να σχεδιάσουμε ένα κύκλωμα βασισμένο στον δημοφιλή μικροεπεξεργαστή 8088, ή για τον ίδιο λόγο ακόμη και τον 8085, τοποθετούσαμε επίσης μνήμη EPROM για την αποθήκευση του προγράμματος, μνήμη RAM για την αποθήκευση μεταβλητών και αποτελεσμάτων διαφόρων πράξεων, καθώς επίσης και μερικές μονάδες εισόδου - εξόδου για την διασύνδεση του συστήματος με το περιβάλλον του, θα είχαμε ουσιαστικά σχηματίσει έναν μικροϋπολογιστή.

Επεκτείνοντας την παραπάνω λογική, όταν όλες οι παραπάνω διακριτές μονάδες που αποτελούν έναν μικροϋπολογιστή τοποθετηθούν μέσα στο σώμα του ίδιου ολοκληρωμένου κυκλώματος, η διάταξη που προκύπτει καλείται μικροελεγκτής. Η εταιρία

Texas Instruments πραγματοποίησε την είσοδο της στον χώρο των μικροελεγκτών, παρουσιάζοντας τον πρώτο μικροελεγκτή της από τον οποίον προέκυψε η οικογένεια των μικροελεγκτών TMS1000. Τα μέλη της σειράς των μικροελεγκτών TMS1000 διέθεταν ικανοποιητικά μεγέθη μνήμης RAM και ROM καθώς επίσης και αρκετές μονάδες εισόδου - εξόδου (I/O) και χρησιμοποιήθηκαν ευρύτατα ως ελεγκτές φούρνων μικροκυμάτων, σε βιομηχανικού τύπου χρονιστές (Industrial Timers) και σε συστήματα αριθμομηχανών.

Σήμερα πλέον έχουν εμφανιστεί πολλές και διαφορετικές οικογένειες μικροελεγκτών, όπως οι μικροελεγκτές της σειράς 8048 και 8051 της Intel, η σειρά 68HC11 της Motorola, η σειρά μικροελεγκτών Z8 της Zilog, οι μικροελεγκτές PIC της Microchip, η σειρά H8 της Hitachi και πρόσφατα η οικογένεια των μικροελεγκτών AVR της εταιρίας Atmel. Μια συγκεκριμένη οικογένεια μικροελεγκτών προσδιορίζει ένα σύνολο πολλών διαφορετικών μικροελεγκτών, οι οποίοι διαθέτουν τον ίδιο κεντρικό πυρήνα αλλά κάθε ένας από αυτούς διατίθεται σε διαφορετική συσκευασία, περιλαμβάνει διαφορετικές περιφερειακές μονάδες, λειτουργεί σε διαφορετικές ταχύτητες, κ.λ.π.

Όπως είδαμε οι ορισμοί των εννοιών μικροελεγκτής, μικροεπεξεργαστής και μικροϋπολογιστής είναι ακριβείς και διακριτοί μεταξύ τους, ωστόσο αποτελεί κοινοτυπία πλέον η χρήση των παραπάνω όρων χωρίς να γίνεται πάντα ο σωστός διαχωρισμός. Έτσι η ασάφεια που επικρατεί σχετικά με τους όρους αυτούς αποτελεί μια πραγματικότητα την οποία μάλλον οφείλουμε να δεχθούμε. Σχετικά με την χρησιμοποιούμενη ορολογία, ο όρος μικροελεγκτής αναφέρεται σε ένα ολοκληρωμένο κύκλωμα με ενσωματωμένες περιφερειακές μονάδες (θύρες, χρονιστές, σειριακές θύρες επικοινωνίας, κ.λ.π.) και μονάδες μνήμης, εκτός από την κεντρική μονάδα επεξεργασίας (CPU).Το αντιπροσωπευτικό μοντέλο των μικροελεγκτών AVR, με ενσωματωμένη μνήμη προγράμματος και δεδομένων, θύρες εισόδου - εξόδου, χρονιστές και σειριακές θύρες, αποτελεί έναν μικροελεγκτή με την έννοια που δόθηκε πιο πάνω.

ΟΙ ΜΙΚΡΟΕΛΕΓΚΤΕΣ AVR ΤΗΣ ATMEL

Κατά την διαδικασία ανάπτυξης μιας εφαρμογής που βασίζεται σε κάποια συγκεκριμένη προδιαγραφή, η ανάγκη υλοποίησης με χρήση διακριτών ψηφιακών κυκλωμάτων, η ολοκληρωμένων κυκλωμάτων PLD ή ακόμα και μικροελεγκτών, προσδιορίζεται πάντα από τον ίδιο τον σχεδιαστή της εφαρμογής. Ωστόσο είναι γεγονός

ότι πάρα πολλές εφαρμογές μπορούν να υλοποιηθούν αποτελεσματικά με χρήση κάποιου μικροελεγκτή, ενώ σε αρκετές από αυτές μπορούμε να εκμεταλλευτούμε τα σημαντικά πλεονεκτήματα των μικροελεγκτών AVR που περιγράφονται περιληπτικά στη συνέχεια. Θα αναφερθούμε με περισσότερες λεπτομέρειες στα χαρακτηριστικά των μικροελεγκτών AVR, αλλά στο σημείο αυτό θεωρείται χρήσιμη μια σύντομη περιγραφή των κυριότερων ιδιοτήτων των μικροελεγκτών αυτών. Έτσι συνοπτικά, οι μικροελεγκτές AVR της Atmel παρουσιάζουν τα ακόλουθα χαρακτηριστικά:

1. Συνδυάζουν την αρχιτεκτονική Risc με ως επί το πλείστον σταθερού μήκους εντολές, διαδικασίες αποθήκευσης - φόρτωσης στην μνήμη και 32 καταχωρητές γενικής χρήσης.
2. Μηχανισμό συνεχούς διοχέτευσης εντολών (instruction pipeline) σε δύο στάδια, που επιταχύνει σημαντικά την διαδικασία εκτέλεσης.
3. Οι περισσότερες από τις εντολές που περιλαμβάνει το ρεπερτόριο τους εκτελούνται στην διάρκεια μιας περιόδου του κεντρικού ρολογιού.
4. Λειτουργούν σε συχνότητες χρονισμού μέχρι τους 20 MHz.
5. Διαθέτουν μεγάλη ποικιλία σε ότι αφορά ενσωματωμένες περιφερειακές μονάδες όπως, ψηφιακές εισόδους - εξόδους (I/O), μετατροπείς αναλογικού σήματος σε ψηφιακό ή ADC, μνήμη τύπου EEPROM, χρονιστές, μονάδες ασύγχρονης σειριακής επικοινωνίας ή UART (Universal Asynchronous Receiver Transmitter), ρολόγια πραγματικού χρόνου (RTC ή Real Time Clock), μονάδες διαμόρφωσης εύρους παλμών (PWM ή Pulse Width Modulation), κ.λ.π.
6. Ενσωματωμένες μνήμες προγράμματος και δεδομένων.
7. Δυνατότητα προγραμματισμού εντός του συστήματος (ISP ή In-System Programmable)
8. Διατίθενται σε συσκευασίες των 8 έως 64 ακροδεκτών οπότε κρίνονται κατάλληλοι για έναν μεγάλο αριθμό διαφορετικών εφαρμογών.
9. Είναι περίπου 12 φορές ταχύτεροι και πιο αποδοτικοί σε σχέση με τους ελεγκτές κλασικής αρχιτεκτονικής CISC (Complex Instruction Set Computer).
10. Ευρεία περιοχή τάσεων λειτουργίας από 2.7 V έως 6.0 V.
11. Η σχετικά απλή αρχιτεκτονική τους δίνει το πλεονέκτημα του μικρού σε απαίτηση κύκλου εκμάθησης στους αρχαρίους.

ΤΑ ΣΥΣΤΑΤΙΚΑ ΜΕΡΗ ΕΝΟΣ ΜΙΚΡΟΕΛΕΓΚΤΗ

1. **Μονάδα κεντρικής επεξεργασίας:** Η μονάδα κεντρικής επεξεργασίας ή Central Processing Unit (CPU), αποτελεί την "καρδιά" ενός μικροελεγκτή. Εκτελεί ανάκληση δεδομένων (fetch) από την μνήμη προγράμματος υπό μορφή εντολών, αποκωδικοποιεί τις εντολές αυτές και στη συνέχεια τις εκτελεί. Η μονάδα CPU αποτελείται από καταχωρητές (registers), την αριθμητική λογική μονάδα (Arithmetic Logic Unit ή ALU), τον αποκωδικοποιητή εντολών (instruction decoder) και διάφορα κυκλώματα ελέγχου.
2. **Μνήμη προγράμματος:** Στην Μνήμη Προγράμματος αποθηκεύονται οι εντολές που σχηματίζουν τον κορμό του προγράμματος. Για την χρήση μεγαλύτερων σε μήκος προγραμμάτων, το είδος της μνήμης αυτής μπορεί αν χωριστεί σε εσωτερική μνήμη προγράμματος και εξωτερική μνήμη προγράμματος, όπως συμβαίνει σε μερικούς τύπους ελεγκτών. Η μνήμη προγράμματος είναι μια μη-πτητική μνήμη {non-volatile memory) και μπορούμε να την συναντήσουμε σε διάφορους τύπους όπως αυτόν την EEPROM (Ηλεκτρικά Επαναπρογραμματιζόμενη Μνήμη Ανάγνωσης Μόνο ή Electrically Erasable Read Only Memory), την EPROM {Επαναπρογραμματιζόμενη Μνήμη Ανάγνωσης Μόνο ή Erasable Read Only Memory), την Μνήμη Ταχείας Αποθήκευσης {Flash Memory), την Μνήμη ROM τύπου Μάσκας {Mask ROM) και την Μνήμη Μη Αναστρέψιμου Προγραμματισμού {On-Time Programmable ή OTP).
3. **Μνήμη RAM:** Η Μνήμη Τυχαίας Προσπέλασης {Random Access Memoiy ή RAM, αποτελεί την μνήμη δεδομένων του ελεγκτή, δηλαδή χρησιμοποιείται από τον ελεγκτή για την αποθήκευση δεδομένων. Η CPU χρησιμοποιεί την μνήμη RAM για την αποθήκευση μεταβλητών καθώς επίσης και την λεγόμενη Στοίβα (ή Stack). Η στοίβα χρησιμοποιείται από την CPU για την προσωρινή αποθήκευση των λεγόμενων διευθύνσεων επιστροφής, με σκοπό να συνεχίσει την εκτέλεση του προγράμματος που είχε διακοπεί για την εξυπηρέτηση κάποιας υπορουτίνας {Subroutine) ή κάποιας ρουτίνας διακοπής {Interrupt routine).
4. **Ταλαντωτής Χρονισμού:** Ο μικροελεγκτής εκτελεί ένα πρόγραμμα όπως αυτό εμφανίζεται μέσα στην μνήμη προγράμματος, με έναν καθορισμένο ρυθμό. Ο ρυθμός αυτός καθορίζεται από την συχνότητα λειτουργίας του ταλαντωτή χρονισμού. Ο Ταλαντωτής Χρονισμού (Clock Oscillator) μπορεί να είναι ένας εσωτερικός ταλαντωτής τύπου RC, ή ένας ταλαντωτής που υλοποιείται με κάποιο εξωτερικό στοιχείο χρονισμού, όπως για παράδειγμα ένας κρύσταλλο χαλαζία (Quartz), ένα

κύκλωμα συντονισμού LC ή ακόμα και ένα απλό κύκλωμα RC. Η λειτουργία του ταλαντωτή ξεκινά σχεδόν αμέσως μετά την εφαρμογή της τάσης τροφοδοσίας.

5. **Σύστημα επανατοποθέτησης και Κύκλωμα ανίχνευσης βυθίσεων τάσης:**

Το Κύκλωμα. Επανατοποθέτησης ή μηδενισμού ή απλά Reset, που διαθέτει ένας μικροελεγκτής , εξασφαλίζει το γεγονός ότι όλες οι εσωτερικές μονάδες και τα κυκλώματα ελέγχου του μικροελεγκτή θα ξεκινήσουν να λειτουργούν κατά την εφαρμογή της τροφοδοσίας, από κάποια προκαθορισμένη αρχική κατάσταση ενώ όλοι οι καταχωρητές του συστήματος βρίσκονται σε κατάλληλες αρχικές τιμές. Από την άλλη πλευρά, το κύκλωμα ανίχνευσης βύθισης της τάσης τροφοδοσίας (brownout detector), είναι ένα επίσης εσωτερικό κύκλωμα ελέγχου το οποίο παρακολουθεί συνεχώς το επίπεδο της τάσης τροφοδοσίας και εφόσον ανιχνευτεί κάποια στιγμιαία βύθιση στην τάση αυτή, τότε αυτόματα θέτει τον μικροελεγκτή σε λειτουργία επανατοποθέτησης, έτσι ώστε να προστατευτούν τα περιεχόμενα των καταχωρητών και της μνήμης από πιθανή καταστροφή ή αλλοίωση, πράγμα που θα οδηγούσε τον μικροελεγκτή σε εσφαλμένη λειτουργία.

6. **Σειριακή θύρα επικοινωνίας:** Ένα από τα πλέον εύχρηστα συστατικά ενός μικροελεγκτή, αποτελεί η σειριακή θύρα επικοινωνίας. Η θύρα αυτή χρησιμοποιείται για την επικοινωνία του ελεγκτή με διάφορες εξωτερικές διατάξεις υπό την μορφή σειριακής μετάδοσης δεδομένων. Η θύρα αυτή μπορεί να λειτουργήσει σε οποιαδήποτε ταχύτητα μετάδοσης δεδομένων τυχόν απαιτηθεί. Η λειτουργία της βασίζεται στο ότι λαμβάνει δεδομένα από τον μικροελεγκτή, τα οποία ολισθαίνει προς την έξοδο υπό μορφή ενός δυαδικού ψηφίου (bit) τη φορά. Εντελώς παρόμοια, λαμβάνει δεδομένα από την αντίστοιχη είσοδο της και πάλι με τη μορφή ενός bit τη φορά, σχηματίζοντας έτσι με 8 τέτοια bits, μια λέξη του 1 byte, την οποία και αντιγράφει στο εσωτερικό του ελεγκτή. Οι σειριακές θύρες απαντώνται σε δύο τύπους, την ασύγχρονη σειριακή θύρα και την σύγχρονη σειριακή θύρα. Για την λειτουργία μιας σύγχρονης σειριακής θύρας απαιτείται και η παρουσία ενός πρόσθετου σήματος συγχρονισμού (clock), αντίθετα με την ασύγχρονη σειριακή θύρα στην οποία δεν απαιτείται η ύπαρξη ενός τέτοιου σήματος, διότι οι απαραίτητες πληροφορίες συγχρονισμού και χρονισμού γενικότερα, ενσωματώνονται στο σύνολο των δεδομένων που μεταδίδονται σειριακά με τη μορφή της διάρκειας εμφάνισης των εκάστοτε bits που αποτελούν την πληροφορία, καθώς επίσης και με τη χρήση πρόσθετων bits με τα οποία σηματοδοτείται η έναρξη και η παύση μιας συγκεκριμένης μετάδοσης..

7. **Ψηφιακή θύρα εισόδου-εξόδου:** Ο μικροελεγκτής χρησιμοποιεί τις ψηφιακές θύρες εισόδου-εξόδου με σκοπό την ανταλλαγή δεδομένων από και προς το εξωτερικό περιβάλλον. Σε σύγκριση με μια σειριακή θύρα, με την οποία τα δε δομένα ανταλλάσσονται υπό μορφή συρμού με εκπομπή ενός bit τη φορά, η ψηφιακή θύρα εισόδου—εξόδου ανταλλάσσει δεδομένα υπό τη μορφή ομάδων των 8 bits, ή διαφορετικά, του 1 byte.

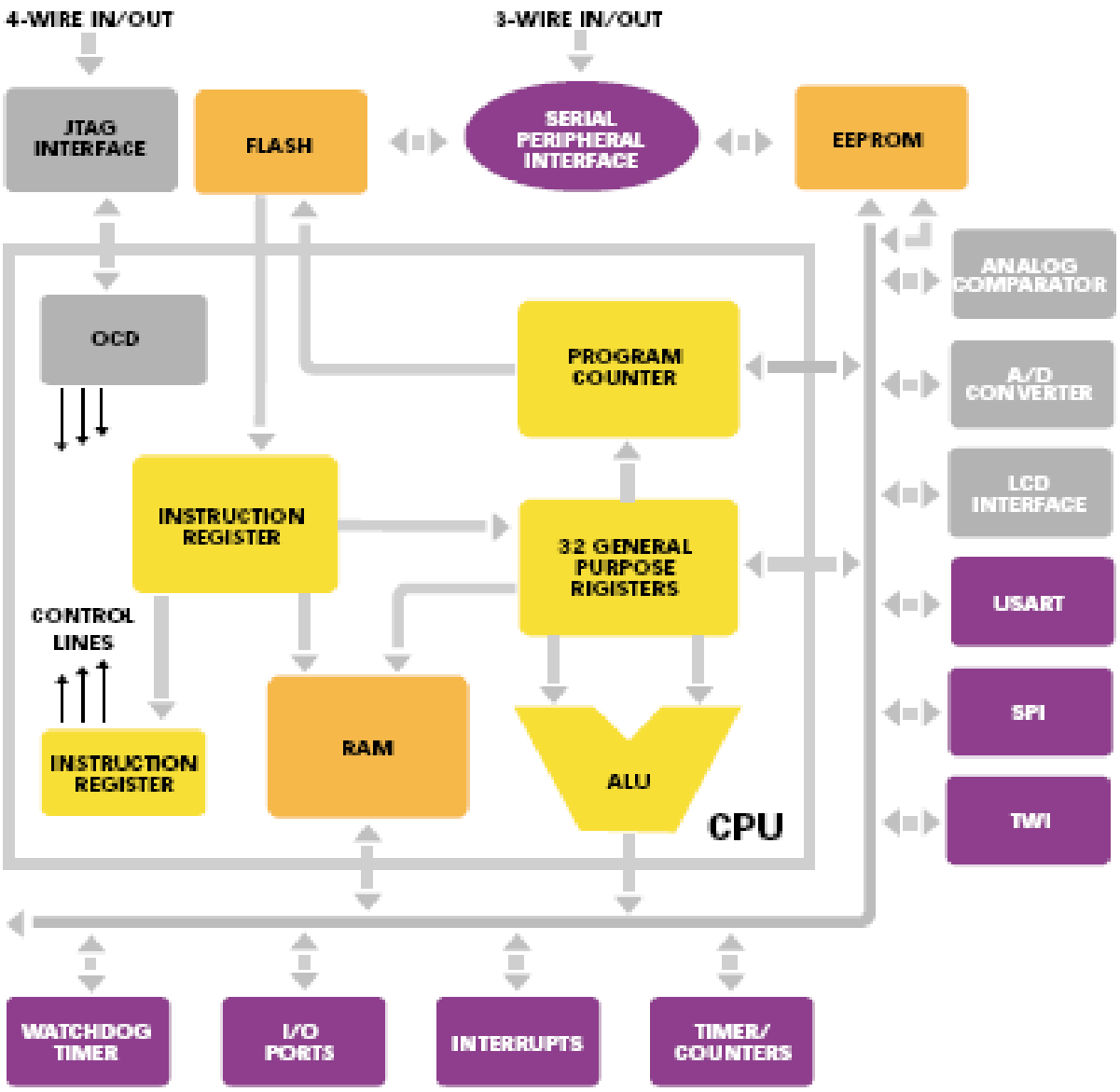
8. **Αναλογική θύρα εισόδου-εξόδου:** Γενικά, μπορούμε να έχουμε αναλογικές εισόδους χρτ\αψοποιώντας Μετατροπείς Ανα?.ογικού Σήματος σε Ψηφιακό (Analog to Digital Converter ή ADC). Ένας τυπικός μικροελεγκτής μπορεί να διαθέτει μια ενσωματωμένη μονάδα μετατροπής ADC ή ακόμα και σε μερικές περιπτώ σεις, έναν απλό ενσωματωμένο αναλογικό συγκριτή, ο οποίος χρησιμοποιείται μαζί με κατάλληλο λογισμικό έτσι ώστε να μπορεί να εκτελεί μετατροπές ανα λογικού σε ψηφιακό. Οι μετατροπείς αναλογικού σήματος σε ψηφιακό χρησιμο ποιούνται για την ανάγνωση δεδομένων από αισθητήρες όπως για παράδειγμα, αισθητήρες πίεσης και θερμοκρασίας. Οι αισθητήρες αυτοί συνήθως παράγουν μια τάση η οποία είναι ανάλογη της μετρούμενης φυσικής παραμέτρου. Επίσης, μπορούμε να έχουμε και αναλογικές εξόδους, χρησιμοποιώντας κάποιες μονά δες οι οποίες καλούνται, Μετατροπείς Ψηφιακού Σήματος σε Αναλογικό (Digital to Analog Converter ή DAC). Ακόμη, οι περισσότεροι μικροελεγκτές είναι εφο διασμένοι με Διαμορφωτές Εύρους Παλμών (Pulse Width Modulators ή PWM), με τη βοήθεια των οποίων μπορούμε να λάβουμε αναλογικές τάσεις μέσα από κατάλληλα φίλτρα τύπου RC. Οι μετατροπείς ψηφιακού σήματος σε αναλογικό χρησιμοποιούνται για την οδήγηση κινητήρων, ειδικών μονάδων απεικόνισης (όπως οι παλαιότερες οθόνες με μπάρα από LED, γνωστές και με την ονομασία VU-Meter), για την αναπαραγωγή σημάτων ήχου ή μουσικής γενικότερα

9. **Χρονιστής:** Ένας Χρονιστής (Timer) χρησιμοποιείται από τον μικροελεγκτή για τον χρονισμό ή και την σηματοδότηση διαφόρων γεγονότων, για παράδειγμα, είναι πιθανό να επιθυμούμε να αποστείλουμε δεδομένα σε μια εξωτερική οθόνη με έναν συγκεκριμένο ρυθμό. Ο χρονιστής χρησιμοποιείται από τον μικροελε γκτή για να παράγει αυτόν τον ζητούμενο ρυθμό. Ένας χρονιστής μπορεί να χρησιμοποιηθεί επίσης και για την καταμέτρηση γεγονότων, τα οποία μπορούν

να είναι είτε εσωτερικά είτε εξωτερικά. Σε μια τέτοια περίπτωση, ο χρονιστής καλείται και απλά, Μετρητής (Counter).

9. Χρονιστής επιτήρησης: Ένας χρονιστής ειδικού σκοπού, τον οποίον συνα ντούμε συχνά στους σύγχρονους μικρό ελεγκτές, είναι και ο Χρονιστής Επιτήρη σης (WatchDog Timer **ή** WDT). Ο χρονιστής αυτός χρησιμοποιείται συνήθως για την αποφυγή της πιθανής κατάρρευσης του συστήματος (crash). Η λειτουρ γία του χρονιστη επιτήρησης έχει ως εξής. Από την στιγμή που θα ενεργοποιη θεί (ή όπως λέγεται, "θα οπλιστεί"), λειτουργεί αυξανόμενα ένας εσωτερικός μετρητής σε κάποιον συγκεκριμένο ρυθμό. Αν το Πρόγραμμα χρήσης δεν μηδε νίσει (ή επαναθέσει) τον μετρητή αυτόν, τότε κάποια στιγμή θα επέλθει η λεγόμενη υπερχείλιση (Overflow) του παραπάνω μετρητή και θα επανατοποθετηθεί ο μικροελεγκτής (λειτουργία reset). Έτσι όταν χρησιμοποιείται ο χρονιστής επιτήρησης, το πρόγραμμα χρήσης θα πρέπει να είναι εφοδιασμένο με κατάλληλες εντολές, έτσι ώστε σε τακτά χρονικά διαστήματα να μπορεί να μηδενίζει τον χρονιστή WDT, πληροφορώντας τον έτσι ότι το σύστημα εργάζεται και δεν έχει καταρρεύσει. Η λογική αυτής της τεχνικής ελέγχου στηρίζεται στην υπόθεση, ότι αν το πρόγραμμα χρήσης δεν μηδενίσει τον χρονιστή WDT, αυτό πιθανότα τα σημαίνει ότι το πρόγραμμα έχει αποτύχει σε κάποια προσπάθεια του είτε εξαιτίας πιθανής κατάρρευσης, ή γενικότερα κάποιας απρόβλεπτης συμπεριφοράς, οπότε είναι προτιμότερο να εκκινήσει διαδικασία επανατοποθέτησης.

10. Ρολόι πραγματικού χρόνου: Ένας ακόμη ειδικού σκοπού χρονιστής είναι και το λεγόμενο Ρολόι Πραγματικού Χρόνου (Real Time Clock ή RTC), του οποίου σκοπός είναι η μέτρηση και η διατήρηση της τρέχουσας ώρας της ημέρας, της ημερομηνίας, κ.λ.π. Μπορεί να χρησιμοποιηθεί για την σηματοδότηση συγκε κριμένων γεγονότων με γνώμονα την τρέχουσα ώρα.



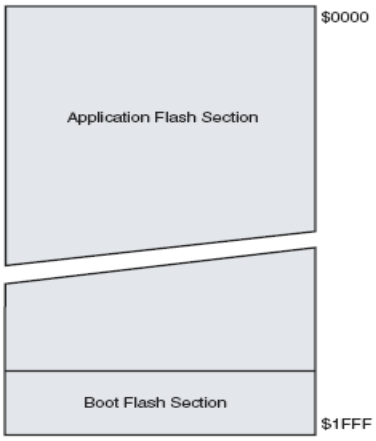
τυπικό διάγραμμα ενός μικροελεγκτή

Η ΜΝΗΜΗ

Η λειτουργία της μνήμης των μικροελεγκτών AVR της ATMEL είναι βασισμένη στο πρότυπο του HARVARD στο οποίο το μέρος της μνήμης είναι χωρισμένο σε διάφορα τμήματα έτσι ώστε να επιτυγχάνεται γρήγορη πρόσβαση σε αυτή καθώς και να παρουσιάζει αυξημένη χωρητικότητα. Στην συνέχεια θα εξετάσουμε την μνήμη ενός μεσαίου μεγέθους μικροελεγκτή ,όπως ο ATMEGA16.

ΜΝΗΜΗ FLASH

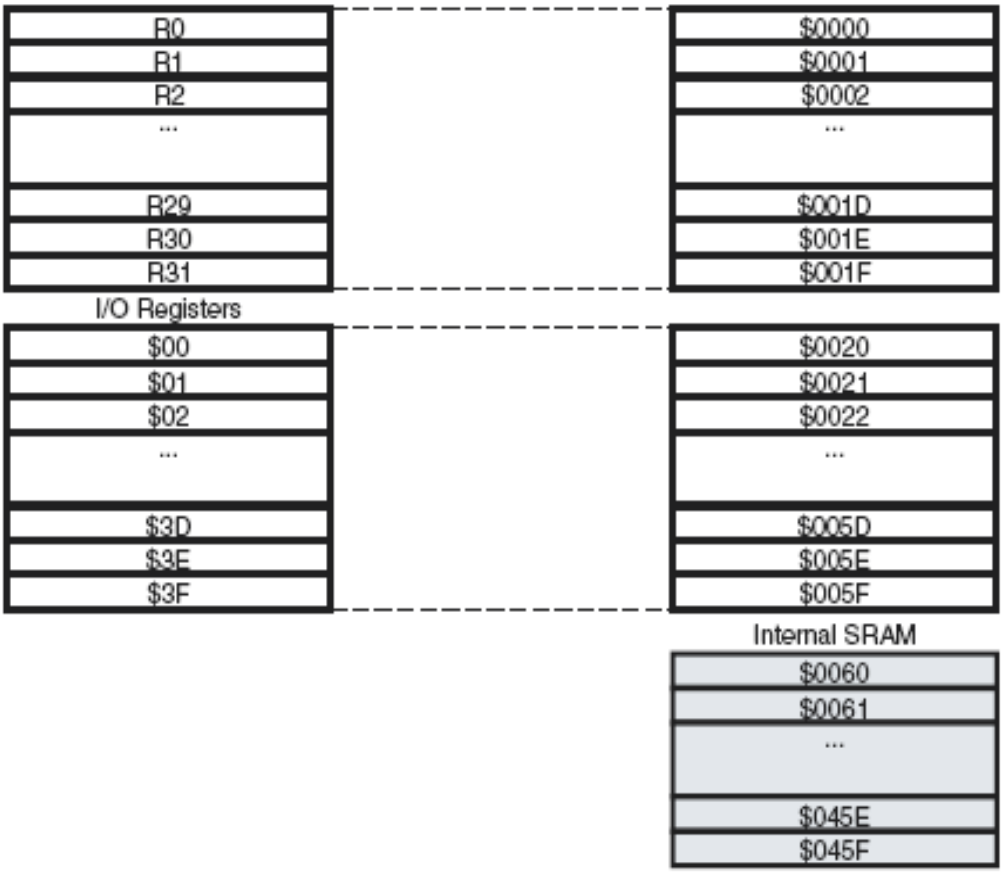
Η μνήμη flash ξεκινάει από την διεύθυνση 0x0000 και το τέλος εξαρτάται από τον κάθε μικροελεγκτή(έως 0x1FFF στον mega16). Η μνήμη flash είναι μια μη πτητική μνήμη και χρησιμοποιείται για να αποθηκεύει τον εκτελέσιμο κώδικα και τις σταθερές μεταβλητές ενός προγράμματος ,επειδή πρέπει να μείνουν στην μνήμη του μικροελεγκτή ακόμα και αν η τροφοδοσία σταματήσει. Το μήκος των λέξεων της μνήμης flash είναι 16bit.Αν και η μνήμη flash μπορεί να προγραμματίζεται ξανά και ξανά, εν τούτοις δεν υπάρχει πρόσβαση εγγραφής σε αυτήν από το πρόγραμμα του μικροελεγκτή. Αυτό σημαίνει πως πρέπει να προγραμματιστεί εξωτερικά. Έτσι λοιπόν από την πλευρά του προγραμματιστή η μνήμη flash συμπεριφέρεται σαν μια μνήμη ROM(read only memory).



← εύρος 16 bit →

Η ΜΝΗΜΗ ΔΕΔΟΜΕΝΩΝ

Η μνήμη δεδομένων (data memory) ουσιαστικά περιέχει τρεις διαφορετικές περιοχές μνήμης ανάγνωσης-εγγραφής. Η χαμηλότερη περιοχή περιέχει τους 32 καταχωρητές γενικής χρήσης. Ακολουθούν 64 καταχωρητές εισόδου –εξόδου(I/O registers)και στην συνέχεια ακολουθεί η περιοχή μνήμης SRAM.



ΚΑΤΑΧΩΡΗΤΕΣ

Οι επεξεργαστές των υπολογιστών και των μικροελεγκτών διαθέτουν κάποιες συσκευές αποθήκευσης οι οποίες ονομάζονται καταχωρητές. Η αποθήκευση σε αυτούς

γίνεται προσωρινά για μικρό χρονικό διάστημα ,ετσι ώστε ο επεξεργαστής να κάνει τις λειτουργίες που χρειάζεται.

ΚΑΤΑΧΩΡΗΤΕΣ ΓΕΝΙΚΗΣ ΧΡΗΣΗΣ

Οι καταχωρητές γενικής χρήσης καταλαμβάνουν τις 32 χαμηλότερες θέσεις στην μνήμη δεδομένων (όπως φαίνεται και στο σχήμα 2.2). Αυτοί οι καταχωρητές χρησιμοποιούνται για να αποθηκεύουν προσωρινά δεδομένα ,αποτελέσματα πράξεων ,τοπικές μεταβλητές και μερικές φορές να χρησιμοποιούνται για έμμεση διευθυνσιοδότηση. Με λίγα λόγια ο επεξεργαστής χρησιμοποιεί αυτούς τους 32 καταχωρητές κατά την διάρκεια εκτέλεσης του προγράμματος. Η χρήση των 32 καταχωρητών ελέγχεται από τον C compiler και στην ουσία δεν απασχολεί τον προγραμματιστή (εκτός αν χρησιμοποιουμε assembly)

ΚΑΤΑΧΩΡΗΤΕΣ ΕΙΣΟΔΟΥ-ΕΞΟΔΟΥ

Στον atmega16 οι καταχωρητές εισόδου-εξόδου καταλαμβάνουν τις επόμενες 64 θέσεις μνήμης στην μνήμη δεδομένων. Ο καθένας από αυτούς τους καταχωρητές παρέχει πρόσβαση στους καταχωρητές ελέγχου, ή τα δεδομένα από τις περιφερειακές συσκευές που εμπεριέχονται στον μικροελεγκτή. Κάθε καταχωρητής εισόδου-εξόδου έχει όνομα, διεύθυνση I/O, και μια διεύθυνση SRAM.

I/O όνομα καταχωρητή	I/O διεύθυνση καταχωρητη	Διεύθυνση SRAM	Περιγραφή
DDRA	0X1A	0X3A	καταχ. κατεύθυνσης portA
DDRB	0X17	0X37	καταχ. κατεύθυνσης PortB
PORTA	0X1B	0X3B	καταχ. δεδομένων portA
SREG	0X3F	0X5F	Καταχ. κατάστασης(status)

Αντιστοιχίζοντας ένα όνομα σε έναν καταχωρητή, είναι πιο εύκολο για τον προγραμματιστή να θυμάται το όνομα του, παρά την διεύθυνση του. Σε έναν μεταγλωττιστή C δεν μας απασχολεί ποιο όνομα αντιστοιχεί σε ποια διεύθυνση. Αυτή την δουλειά την αναλαμβάνει ο μεταγλωττιστής. Κάθε πρόγραμμα περιέχει μια επικεφαλίδα **#include** (header file) η οποία αντιστοιχίζει τα ονόματα με τις διευθύνσεις κάθε καταχωρητή.

```
#include <mega16.h>

main()

{

        DDRA=0XFF;              //όλα τα bit της porta ως είξοδοι

        DDRB=0X00;              // όλα τα bit της portb ως είσοδοι

        while(1)

        {

        PORTA=PINB          //διαβασε την portb και γράψε στην porta

        }

}
```

Το παραπάνω πρόγραμμα διαβάζει τα pins της portb χρησιμοποιώντας τον καταχωρητή PINB I/O και γράφει το αποτέλεσμα στην porta.Ο μεταγλωττιστής παίρνει τις διευθύνσεις των καταχωρητών και τις αντιστοιχεί ονομαστικά (DDRA, DDRB, PORTA, PINB) όλα αυτά εμπεριέχονται στο αρχείο #include και στην συγκεκριμένη περίπτωση αναφέρονται στον μικροελεγκτή atmega16.

Η ΜΝΗΜΗ SRAM

Η μνήμη sram χρησιμοποιείται για να αποθηκεύει μεταβλητές οι οποίες δεν χωράνε στους καταχωρητές καθώς και για να αποθηκεύουν την στοίβα του επεξεργαστή (stack). Τα δεδομένα συνήθως αποθηκεύονται ξεκινώντας από την διεύθυνση 0x0060 και σταματούν στην διεύθυνση 0x045F για τον μικροελεγκτή atmega16.

Η ΜΝΗΜΗ EEPROM

Η μνήμη eeprom είναι μια μη-πτητική μνήμη. Δηλαδή τα δεδομένα με την διακοπή της τροφοδοσίας διατηρούνται αναλλοίωτα μέχρι την επόμενη προσπέλαση της. Η eeprom ξεκινάει από την διεύθυνση 0x000 και στον mega16 φτάνει την 0x1FF. Βέβαια το μέγεθος της eeprom εξαρτάται από τον εκάστοτε μικροελεγκτή. Στην eeprom έχουμε την δυνατότητα να εγγράψουμε δεδομένα, καθώς και να ανακτήσουμε. Θα ήταν ιδανικό να μπορούσαμε να την χρησιμοποιήσουμε για να αποθηκεύουμε τις μεταβλητές μας εκεί συνεχώς,δηλαδή να την χρησιμοποιούσαμε ως sram.Όμως ο μεγάλος χρόνος που απαιτεί για να κάνει μια εγγραφή (περίπου 1ms για να εγγράψει 1 byte) καθώς και ο περιορισμένος χρόνος ζωής της (100.000 κύκλοι εγγραφής) την κάνει κατάλληλη μόνο για μεταβλητές και τιμές που χρειάζονται να μείνουν στην μνήμη του μικροελεγκτή ακόμα και αν διακοπεί η τροφοδοσία.

ΔΙΑΚΟΠΕΣ (INTERRUPTS)

Μια Διακοπή {Interrupt) αποτελεί έναν μηχανισμό ροής ελέγχου ο οποίος απαντά-ται στους περισσότερους ελεγκτές. Σε ένα σύστημα επεξεργαστή το οποίο αλληλεπιδρά με το περιβάλλον του, λαμβάνουν χώρα πολύ συχνά διάφορες ενέργειες ασύγχρονης φύσης, όπως για παράδειγμα, το πάτημα ενός διακόπτη την στιγμή που καταφθάνει κάποια πληροφορία στην θύρα σειριακής επικοινωνίας. Είναι φυσικά αδύνατο για το σύστημα του επεξεργαστή, να εξετάζει ταυτόχρονα όλων των ειδών τις πιθανές ενέργειες ελέγχοντας συνεχώς τις περιφερειακές μονάδες για τυχόν άφιξη νέων δεδομένων. Θεωρείται προτιμότερο να υπάρχει η δυνατότητα στις διάφορες περιφερειακές μονάδες να "ανακοινώνουν" κατά κάποιον τρόπο, την άφιξη νέων δεδομένων. Τον ρόλο αυτόν

αναλαμβάνει ο μηχανισμός των διακοπών. Μια περιφε-ρειακή μονάδα είναι σε θέση να "διακόψει" την ροή της λειτουργίας του κύριου προγράμματος, οπότε ο επεξεργαστής αποκόπτεται από την κανονική του λειτουργία με σκοπό να εξετάσει την πηγή που προκάλεσε την συγκεκριμένη διακοπή και να ενεργήσει ανάλογα. Αφού εκτελεστούν οι απαραίτητες ενέργειες που έχουν σχέση με την διακοπή, ο έλεγχος της ροής λειτουργίας επιστρέφει στο κύριο πρόγραμμα και συγκεκριμένα στο σημείο όπου εμφανίστηκε η διακοπή. Ένα τμήμα προγράμματος που αφορά την εξυπηρέτηση μιας διακοπής είναι ουσιαστικά μια υπο-ρουτίνα με την διαφορά ότι η εκτέλεση αυτής της ρουτίνας εξυπηρέτησης διακοπής, δε προβλέπεται από τον επεξεργαστή να συμβεί σε κάποια συγκεκριμένη χρονική στιγμή.

Οι μικροελεγκτές της οικογένειας AVR διαθέτουν ένα σύστημα διακοπών με αρ-κετά πλούσια δομή. Η δυνατότητα ενεργοποίησης διακοπών έχει δοθεί στις περισσότερες από τις διαθέσιμες περιφερειακές μονάδες των μικροελεγκτών έτσι ώστε να μην είναι απαραίτητο ο ελεγκτής να εξετάζει κάθε στιγμή διαδοχικά όλες τις μονάδες αυτές.Η ακολουθία των γεγονότων που συνοδεύουν την εμφάνιση μιας διακοπής, περιλαμβάνει:

1. Η αντίστοιχη περιφερειακή μονάδα πληροφορεί το σύστημα ότι αιτείται μια διακοπή.
2. Ολοκληρώνεται η εκτέλεση της εντολής που βρίσκονταν σε εξέλιξη κατά την εμφάνιση της αίτησης διακοπής.
3. Η διεύθυνση της επόμενης εντολής του κύριου προγράμματος που διεκόπη, απο-θηκεύεται στην στοίβα του συστήματος (είτε αυτή αντιστοιχεί σε μονάδα υλικού ή τμήμα λογισμικού)
4. Φορτώνεται η διεύθυνση της πρώτης εντολής της ρουτίνας εξυπηρέτησης διακοπής στον μετρητή προγράμματος (PC).
5. Ο επεξεργαστής εκτελεί την Ρουτίνα Εξυπηρέτησης Διακοπής (Interrupt Service Routine ή ISR).
6. Η ολοκλήρωση της εκτέλεσης της ρουτίνας εξυπηρέτησης διακοπής σηματοδοτείται με χρήση της εντολής RETI (Επιστροφή από Ρουτίνα Διακοπής ή RETturn from Interrupt).
7. Ο επεξεργαστής φορτώνει τον μετρητή προγράμματος με την διεύθυνση που είναι αποθηκευμένη στην στοίβα, οπότε συνεχίζεται κανονικά η εκτέλεση του κύριου προγράμματος από το σημείο που εμφανίστηκε η διακοπή.

Στους μικροελεγκτές AVR όλες οι διακοπές έχουν την ίδια προτεραιότητα. Δεν επιτρέπεται μια διακοπή να ξεκινήσει, όταν μια άλλη διακοπή είναι σε εξέλιξη. Στην

περίπτωση που δυο διακοπές θα αιτηθούν ταυτόχρονα, θα εκτελεστεί πρώτα αυτή με το χαμηλότερο αριθμημένο διάνυσμα διεύθυνσης.

Διεύθυνση έναρξης	Νούμερο διανύσματος	Αιτία διακοπής	Περιγραφή
0x0000	1	Reset	Εξωτερ.επανεκινηση,watchdog
0x0002	2	External interrupt 0	Σήμα στο pin INT0
0x0004	3	External interrupt 1	Σήμα στο pin INT1
0x0006	4	Timer 2 compare	Όταν ο συγκριτής του timer2 συμπίμπτει

Οι διακοπές πρέπει να αρχικοποιηθούν πριν γίνουν χρησιμοποιηθούν η γίνουν ενεργές. Η αρχικοποίηση είναι μια διαδικασία 2 βημάτων. Αρχικά πρέπει κάνουμε ενεργές τις διακοπές ,έτσι ώστε να ενεργοποιηθούν και στην συνέχεια να αρχικοποιήσουμε όλες τις ενεργές διακοπές.

Ενεργοποίηση των διακοπών σημαίνει πως πρέπει να θέσουμε την λογική τιμή 1 στον καταχωρητή ελέγχου (control register) στην αντίστοιχη διακοπή που θέλουμε να χρησιμοποιήσουμε. Για παράδειγμα βλέπουμε τον γενικό καταχωρητή ελέγχου διακοπών (GICR)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
INT1	INT0	INT2	-	-	-	-	-

Ο καταχωρητής GICR χρησιμοποιείται για να ενεργοποιήσει εξωτερικές διακοπές. Για παράδειγμα θέτοντας το bit INT0 θα ενεργοποιήσει την διακοπή interrupt 0. Αντίστοιχα θέτοντας το bit INT0 θα ενεργοποιηθεί η διακοπή interrupt 1. Θέτοντας και τα 2 bit INT0 και INT1 θα ενεργοποιηθούν και οι 2 διακοπές.

Για παράδειγμα όταν ένα hardware σήμα εμφανιστεί στο pin INT1,γίνεται η λογική πράξη AND με το αντίστοιχο bit του GICR. Αν το αποτέλεσμα είναι λογικό 1,τότε η διακοπή

επιτρέπεται να εκτελεστεί. Το δεύτερο βήμα είναι να ενεργοποιήσουμε το bit του καταχωρητή κατάστασης (status reg) που αναφέρεται στις ολικές διακοπές (global interrupts). Για να γίνει αυτό στην γλώσσα προγραμματισμού C πρέπει να γράψουμε την εξής γραμμή κώδικα:

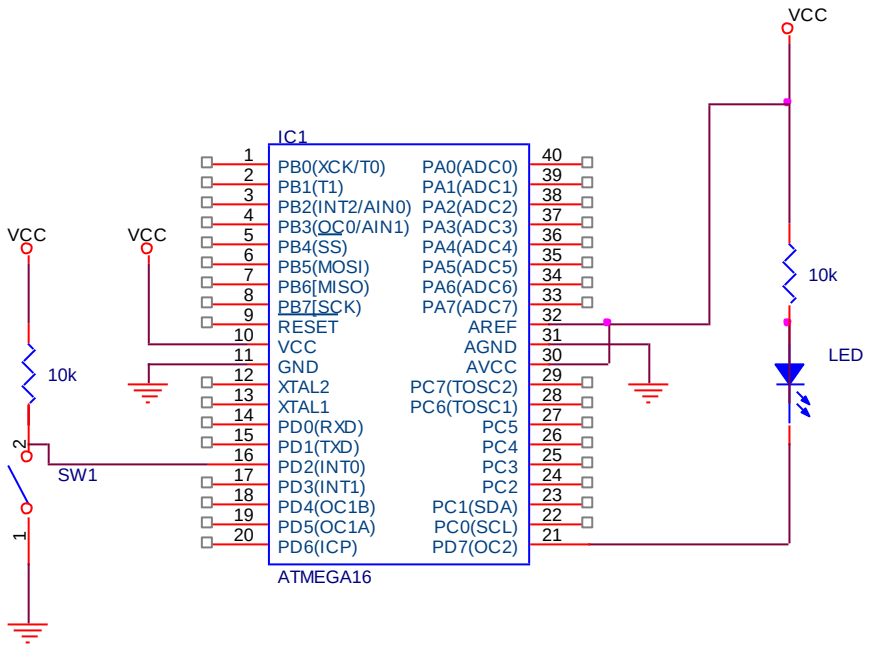
```
#asm("sei")
```

Η παραπάνω γραμμή σημαίνει πως στο σημείο εκείνο του κώδικα θα ενεργοποιηθούν οι διακοπές. Η ψευδοεντολή #asm σημαίνει πως μέσα στο όρισμα θα είναι κώδικας assembly. Η οδηγία που δίνουμε SEI σημαίνει πως θα ενεργοποιηθούν οι διακοπές.

παρακάτω θα δούμε ένα απλό παράδειγμα γραμμένο σε γλώσσα C

```
#include <mega16.h>

 interrupt [EXT_INT0] void ext_int0_isr(void)

        {

        PORTC=PORTC^0x1;   //αναβόσβησε το pin 0 στην πορτα c

        }

void main(void)

{

        DDRC=0x01 ;             //ορισμός της πόρτας c ως έξοδος

        GICR=0x40 ;            //θέσε το bit για ενεργοποίηση της interrupt 0

        MCUCR=0x02 ;        //θέσε το bit ISC01 για ενεργοποιηση διακοπών στο
                        //καθοδικο σημειο του παλμού

        #asm("sei")             //ενεργοποίηση όλων των διακοπών

while(1);

}
```

το πρόγραμμα αναβοσβήνει το led που είναι στην portc στο bit 0 κάθε φορά που το push button που είναι συνδεμένο στο pin INT0 πατιέται. Παρακάτω βλέπουμε το κύκλωμα της εφαρμογής μας.



ΔΙΑΚΟΠΗ ΕΠΑΝΝΕΚΙΝΗΣΗΣ (RESET)

Η επανεκκίνηση αποτελεί μια ειδική διακοπή. Έχει την υψηλότερη προτεραιότητα από όλες τις διακοπές και από οποιονδήποτε κώδικα εκτελείται στον μικροελεγκτή. Οι αιτίες που μπορούν να προκαλέσουν μια διακοπή επανεκκίνησης είναι οι εξής:1) Αν ένα λογικό 0 εμφανιστεί στο pin reset για διάστημα μεγαλύτερο από 50ns, 2)Αν η τάση τροφοδοσίας πέσει κάτω από ένα προκαθορισμένο επιπεδο, τότε ο μικροελεγκτής επανατοποθετείται και το πρόγραμμα ξεκινά από την αρχή για να προστατέψει την λειτουργία του προγράμματος από τυχόν ασταθείς λειτουργίες 3) Αν ο μετρητής watchdog timer μηδενίσει, πράγμα που σημαίνει πως για κάποιο λόγο το πρόγραμμα δεν λειτουργεί σωστά.

WATCHDOG TIMER

Ο watchdog timer είναι μια συσκευή ασφαλείας. Είναι σχεδιασμένο έτσι ώστε να προκαλεί επανεκκίνηση στον μικροελεγκτή, όταν αυτός έχει μπερδευτεί ή εκτελεί οποιονδήποτε άλλον κώδικα από αυτόν που είναι σχεδιασμένος να εκτελεί. Ο watchdog timer είναι ένας μετρητής που προκαλεί επανεκκίνηση όταν αυτός μηδενίσει από την τιμή που μετρούσε. Όταν το πρόγραμμα εκτελείται κανονικά τότε φορτώνουμε η επανεκκινούμε τον watchdog timer έτσι ώστε αυτός να μην μηδενίσει και προκαλέσει reset. Η ουσία του θέματος είναι πως αν το πρόγραμμα κολήσει για κάποιο λόγο, τότε η επαναφορά του μικροελεγκτή θα ξεκινήσει το πρόγραμμα κανονικά. Παρακάτω βλέπουμε τον καταχωρητή ελέγχου του watchdog timer WDTCR

Bit 7	Bit 6	Bit 5	Bit 4	Bit3	Bit 2	Bit 1	Bit 0
			WDTOE	WDE	WDP2	WDP1	WDP0

Bit(s)	Όνομα	Περιγραφή
WDTOE	Watchdog Timer Of Enable	Επιτρέπει την απενεργοποιηση τουWDT
WDE	Watchdog Timer Enable	Ενεργοποιεί τον WDT
WPx	Watchdog Prescaler Bit x	Θέτει τον χρόνο timeout του WDT

Το ρολόι χρονισμού του watchdog timer είναι ξεχωριστό από αυτό του υπόλοιπου συστήματος. Η συχνότητα του εξαρτάται από την τάση τροφοδοσίας του μικροελεγκτή. Αν η τάση τροφοδοσίας είναι 5 volt , η συχνότητα του watchdog timer είναι περίπου 1 MHz. Παρακάτω βλέπουμε τα bit του καταχωρητή WDTCR που αναφέρονται στον προδιαιρέτη που μπορεί να χρησιμοποιηθεί ,ανάλογα με τον επιθυμητό χρόνο επανεκκίνησης του watchdog timer. Επίσης βλέπουμε την εξάρτηση της συχνότητας από την τάση τροφοδοσίας του μικροελεγκτή

WDP2	WDP1	WDP0	Time out (5 volt VCC)	Time out (3 volt VCC)
0	0	0	16ms	16ms
0	0	1	33ms	34ms
0	1	0	65ms	68ms
0	1	1	130ms	140ms
1	0	0	260ms	260ms
1	0	1	520ms	550ms
1	1	0	1000ms	1000ms
1	1	1	2100ms	2200ms

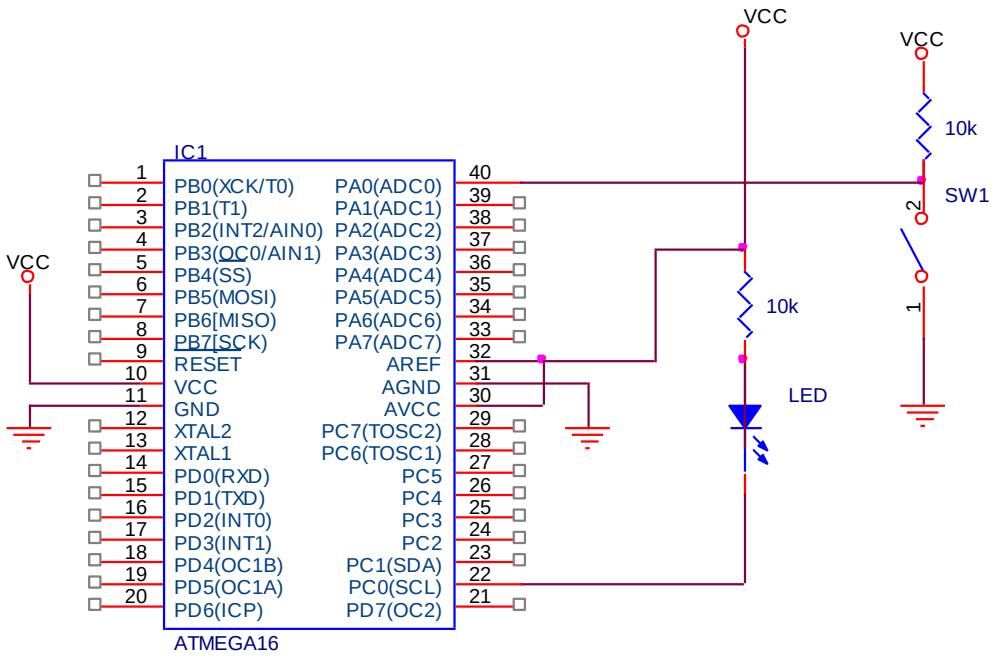
Παρακάτω βλέπουμε ένα παράδειγμα με τον watchdog timer στον μικροελεγκτή mega 16

```
#include <mega16.h>

static unsigned int waiter;

void main(void)

{

        DDRC=0x01;          //θέσε το PORTC.0 για έξοδο

        while (PINA.0==1)   //περίμενε να πατηθεί το μπουτόν

        {

                ;

        }

        WDTCR=0x0b;         //ενεργοποίηση του watchdog timer με χρόνο επανεκκίνησης

                            //130ms
```

```
        while(PINA.0==0)     // για όσο διάστημα το pina.0 είναι πατημένο,μείνε στην
                             //ρουτίνα
        {
            #asm("wdr")'          //επανεκκίνησε τον μετρητή του WDT συνεχώς
            ++waiter ;            //αυξησε την μεταβλητή waiter
            if (waiter ==50000)
            {
                PORTC.0=PORTC.0^1;
                waiter=0;         //μηδένισε την μεταβλητή waiter
            }
        }
}
while(1);
}
```

Όταν το πρόγραμμα ξεκινάει ,παραμένει μέσα στο πρώτο while loop μέχρι να πατηθεί το pushbutton. Όταν πατηθεί το button το πρόγραμμα ενεργοποιεί τον watchdog timer . Σε αυτό το σημείο ο WDT πρέπει να επανατοποθετηθεί πριν υπερχειλίσει και προκαλέσει reset (μετά από 120ms). Για όσο διάστημα το pushbutton είναι πατημένο το πρόγραμμα παραμένει στο δεύτερο while loop. Εκεί ο WDT επανατοποθετείται μέσω της εντολής #asm("wdt");.Αυτή η εντολή είναι σε γλώσσα assembly καθώς δεν υπάρχει αντίστοιχη για την γλώσσα C. Σε αυτό το διάστημα βλέπουμε το led Που είναι συνδεμένο στην portc να αναβοσβήνει. Αν αφήσουμε το pushbutton, τότε το πρόγραμμα θα μεταβεί στο πρώτο while loop.Επειδή ο WDT δεν επανατοποθετείται τώρα θα προκαλεί reset στο σύστημα μετά από 130ms Ως αποτέλεσμα δεν θα παρατηρούμε τίποτα στο παράδειγμα μας.

παρακάτω βλέπουμε το hardware κύκλωμα του παραδείγματός μας.



ΑΝΑΛΟΓΙΚΕΣ ΔΙΑΣΥΝΔΕΣΕΙΣ

Σε αντίθεση με τους μικροελεγκτές που αποτελούν ψηφιακά συστήματα, η φύση λειτουργεί αναλογικά. Οι μικροελεγκτές μπορούν να χειρίζονται αναλογικά σήματα, αφού προηγουμένως τα μετατρέψουν σε ψηφιακά. Ένας μικροελεγκτής AVR εμπεριέχει μία μονάδα μετατροπής αναλογικού σήματος σε ψηφιακό(analog to digital conversion peripheral), καθώς επίσης και μία μονάδα αναλογικής σύγκρισης (analog comparator peripheral). Οι μικροελεγκτές χρησιμοποιούν μετατροπείς ADC για να μετατρέψουν αναλογικές ποσότητες όπως η θερμοκρασία και η τάση σε ψηφιακή μορφή, όπως επίσης για μετατροπή ηχητικού σήματος σε ψηφιακό και πληθώρα άλλον αντίστοιχων εφαρμογών.

Η διαδικασία της ψηφιοποίησης ενός σήματος στηρίζεται στην αναλογία. Το αποτέλεσμα της ψηφιοποίησης ενός σήματος από τον ADC είναι ένας δυαδικός αριθμός 8 η 12 bit, ο οποίος είναι ανάλογος με την τάση πλήρους κλίμακας. Για παράδειγμα έστω πως εφαρμόζουμε 2 volt σε μια είσοδο ενός ADC στην οποία η πλήρη κλίμακα είναι 5 volt. Η ψηφιακή λέξη θα είναι 40% της πλήρης κλίμακας (2/5=0.4). Ένας μετατροπέας των 8 bit θα μας παρέχει εξόδους από 0 μέχρι 2^8-1 η 255 ,ενώ ένας μετατροπέας των 10 bit θα μας παρέχει εξόδους από 0 μέχρι 2^{10}-1 η 1023. Στο παραπάνω παράδειγμα όπου η τάση ήταν 2

volt με πλήρη κλίμακα τα 5 volt, ένας ADC των 8 bit θα διαβάσει το 40% του 255 δηλαδή το 102. Η σχέση τάσης εισόδου με την ψηφιακή τάση εξόδου δίνεται από την σχέση $V_{εισόδου}/V_{πλήρους\ κλίμακας}= x / 2^n-1$, όπου x είναι η ψηφιακή έξοδος και n είναι ο αριθμός των bits. Ένα πολύ σημαντικό μέγεθος που πρέπει να παίρνουμε υπόψιν μας είναι η τάση ανάλυσης η οποία δίνεται από τον τύπο $V_{ανάλυσης}=V_{πλήρης\ κλίμακας} / 2^n-1$. Στο παραπάνω παράδειγμα η τάση ανάλυσης θα είναι $V=5/(2^8-1)=5 /255=20mV$. Αυτό σημαίνει πως στην καλύτερη περίπτωση η διακριτικότητα του ADC θα ανέρχεται στα 20mV.

Η ΜΟΝΑΔΑ ΤΟΥ ΜΕΤΑΤΡΟΠΕΑ ΤΑΣΗΣ (ADC PERIPHERAL)

Η μονάδα ADC έχει την δυνατότητα να παράγει ανάλυση μέχρι 10 bit, καθώς και να λειτουργεί σε ταχύτητες εώς 15ksps(kilo samples per second). Μπορεί να διαβάζει την τάση σε ένα από τα οκτώ διαφορετικά pin εισόδου του μικροελεγκτή κάθε φορά, πράγμα που σημαίνει πως μπορεί χρησιμοποιήσει εως οκτώ διαφορετικές εισόδους διαδοχικά. Τον ADC τον ελέγχουν δυο καταχωρητές. Ο καταχωρητής κατάστασης και ελέγχου του ADC ADCSRA (adc control and status register) και ο καταχωρητής πολύπλεξης ADMUX (adc multiplexer select register), ο οποίος ελέγχει ποια από τις οκτώ πιθανές εισόδους πρόκειται να χρησιμοποιηθεί από τον ADC.

Για την λειτουργία του ο ADC απαιτεί η συχνότητα ρολογιού που θα τρέχει να κυμαίνεται από 50Khz έως 200Khz για να έχει μέγιστη ανάλυση στις τιμές του. Μεγαλύτερες τιμές από 200Khz επιτρέπονται, αλλά η ανάλυση του ADC μειώνεται. Η συχνότητα λειτουργίας του ADC λαμβάνεται με έναν τρόπο παρόμοιο όπως με τον προδιαιρέτη στους χρονιστές. Τα τρία λιγότερο σημαντικά bits του καταχωρητή ADCSRA ελέγχουν την διαίρεση του κεντρικού ρολογιού του συστήματος. Διαιρώντας την συχνότητα του συστήματος με κάποιο από τον συνδιασμό τον τριών bit πρέπει να παίρνουμε χρόνο ανάμεσα στα 50Khz και στα 200 Khz. Παρακάτω βλέπουμε τον καταχωρητή ADCSRA.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

BIT	ΠΕΡΙΓΡΑΦΗ
ADEN	Ενεργοποίηση του ADC. Το θέτουμε λογικό 1 για ενεργοποίηση
ADSC	Έναρξη μετατροπής του ADC.
ADATE	Λειτουργία διαδοχικών μετατροπών
ADIF	Σημαία διακοπής λογο ολοκλήρωσης μετατροπής του ADC
ADIE	Επιτρέπει το interrupt να συμβει λόγο ολοκλήρωσης μιας μετατροπής
ADPS2	Bit ελέγχου του προδιαιρέτη
ADPS1	
ADPS0	

Η μονάδα του ADC είναι πιο αργή από τον επεξεργαστή. Αν λοιπόν ο ADC έκανε συνεχώς μετατροπές θα σπαταλούσε πολύτιμη επεξεργαστική ισχύ της cpu. Έτσι συνήθως ο ADC χρησιμοποιείται με την λειτουργία των διακοπών , ειδικά σε εφαρμογές απαιτητικές για υπολογιστική ισχύ. Παρόλα αυτά μπορούμε να τον λειτουργήσουμε σε free running mode ,όπου θα κάνει μετατροπές συνεχώς όσο πιο γρήγορα είναι δυνατών. Όταν χρησιμοποιούμε τον μικροελεγκτή στην λειτουργία free running είναι απαραίτητο να απενεργοποιήσουμε τις διακοπές στο σύστημά μας, να διαβάσουμε τα δεδομένα του ADC και στην συνέχεια να ενεργοποιήσουμε πάλι τις διακοπές που τυχόν έχουμε μέσα στο πρόγραμμά μας. Αυτή η διαδικασία πρέπει να γίνει για να είμαστε σίγουροι πως τα δεδομένα μας θα είναι σωστά, χωρίς το πρόγραμμα να προσπαθεί να διαβάσει τον ADC, όταν αυτός ανανεώνει τα δεδομένα του. Παρακάτω βλέπουμε τον συνδυασμό των τριών λιγότερο σημαντικών bit του ADCSRA καθώς και την προδιαίρεση που προσφέρουν.

ADPS2	ADPS1	ADPS0	Παράγοντας διαίρεσης
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

συνδυασμός των ADC bit του προδιαιρέτη

Για την αρχικοποίηση του ADC συνήθως ακολουθούνται τα παρακάτω βήματα

1) Θέτουμε τα 3 χαμηλότερα bit του ADCSR στον επιθυμητό χρόνο προδιαίρεσης.

2) θέτουμε το ADIE σε λογικό 1 για την ενεργοποίηση των διακοπών

3) θέτουμε το ADEN σε λογικό 1 για την ενεργοποίηση του ADC.

4) θέτουμε το ADSC για άμεση έναρξη μετατροπής του σήματος εισόδου.

Για έναν λόγο προδιαίρεσης 8, οι παρακάτω γραμμές κώδικα θα αρχικοποιήσουν τον ADC για να διαβάσει μια αναλογική τάση που εφαρμόζεται στο pin ADC2.

```
ADMUX=2;            //διάβασε την αναλογική τάση στο ADC2
ADCSR=0Xcb;         //ενεργοποίηση του ADC,των διακοπών ,η διαίρεση του
                    //ρολογιού /8 και άμεση έναρξη μετατροπών
```

Η παραπάνω αρχικοποίηση θέτει τον ADC σε λειτουργία, θα τον αρχικοποιήσει και θα ξεκινήσει να κάνει μετατροπές στην τάση που εφαρμόζεται στο pin ADC2. Παρακάτω θα αναπτύξουμε μια εφαρμογή η οποία θα ανιχνεύει το επίπεδο τάσης που εφαρμόζεται στην είσοδο ADC3 σε έναν μικροελεγκτή atmega16. Ανάλογα με το επίπεδο της τάσης θα ανάβει και το αντίστοιχο led. Έτσι αν η τάση είναι μεγαλύτερη από 3 Volt θα ανάβει το κόκκινο led, αν η τάση είναι ανάμεσα από 2 Volt και 3 Volt θα ανάβει το πράσινο led και αν η τάση είναι μικρότερη από 2 Volt θα ανάβει το πορτοκαλί led.

```
#include <mega16.h>
#define leds portc
#define red 0b11011111
#define green 0b01111111
#define yellow 0b1011111
Interrupt [ADC_INT] void adc_isr(void)
{
        Unsigned int adc_data;
        adc_data=ADCW;      //ψευδοεντολή του compiler,ενώνει τα 2 byte του ADC
        if (adc_data >(3*1023)/5      //
        {
                Leds=red;
        }
        Else if(adc_data<(2*1023)/5)
        {
                Leds=yellow;
        }
```

```
        Else
        {
                Leds=green;
        }
        ADCSRA=ADCSRA | 0x40;   //ξεκίνα μια νέα μετατροπή
        }
        void main(void)
        DDRC=0xe0;              //τα 3 πιο σημαντικά bit ως έξοδος
        ADMUX=0x3;              //το ADC3 ως έξοδος
        ADCSRA=0Xce;   //ενεργοποίηση του ADC, /64 ,ενεργοποίηση διακοπών
        #asm("sei")    //ενεργοποίηση όλων των διακοπών.
        While(1);
}
```

ΧΡΟΝΙΣΤΕΣ - ΜΕΤΡΗΤΕΣ (TIMER/COUNTERS)

Οι timer/counters είναι από τις πιο κοινές διατάξεις που χρησιμοποιούνται σε έναν μικροελεγκτή. Είναι πολύ ευέλικτες περιφερειακές μονάδες και χρησιμοποιούνται για να μετράνε περίοδους, να βρίσκουν το εύρος ενός παλμού, να μετράνε ταχύτητα, να βρίσκουν την συχνότητα ενός σήματος ή να παρέχουν σήματα χρονισμού.

Οι timer/counters είναι στην ουσία δυαδικοί μετρητές. Όταν χρησιμοποιούνται στην λειτουργία του χρονιστή , οι δυαδικοί μετρητές μετράνε τις περιόδους που εφαρμόζονται στις εισόδους τους ,ενώ στην λειτουργία του μετρητή , μετράνε τους παλμούς που δέχονται στις εισόδους τους.

Οι μικροελεγκτές AVR διαθέτουν μετρητές εύρους 8-bit και 16-bit. Σε κάθε περίπτωση μια αναγκαία ενέργεια που πρέπει να κάνει το πρόγραμμα ,είναι να ελέγχει πότε οι μετρητές φτάνουν την μέγιστη τιμή τους και επανεκκινούνται. Στην περίπτωση των 8-bit μετρητών αυτό συμβαίνει όταν φτάσει την δεκαδική τιμή 255. Έπειτα ο μετρητής επιστρέφει στο 0. Όταν έχουμε μετρητή εύρους 16-bit το ίδιο πράγμα συμβαίνει όταν φτάσει στην δεκαδική τιμή 65535. Το πότε φτάνει ένας μετρητής στην υπερχείλιση του ,πρέπει να το γνωρίζει το πρόγραμμα ,έτσι ώστε να υπάρχει ακρίβεια στα αποτελέσματα που διέπονται από τους χρονιστές.

Μερικοί μικροελεγκτές AVR διαθέτουν δύο χρονιστές των 8-bit (τυπικά τον timer0 και τον timer2) και έναν των 16-bit (τυπικά τον timer1) αν και αυτή η διάταξη των μετρητών εξαρτάται από τον εκάστοτε μικροελεγκτή που χρησιμοποιούμε. Στην συνέχεια θα δούμε τους προδιαιρέτες χρόνου που έχουν οι χρονιστές , καθώς και παραδείγματα με τις πιο συνηθισμένες χρήσεις των μετρητών.

ΟΙ ΠΡΟΔΙΑΙΡΕΤΕΣ ΤΩΝ ΧΡΟΝΙΣΤΩΝ

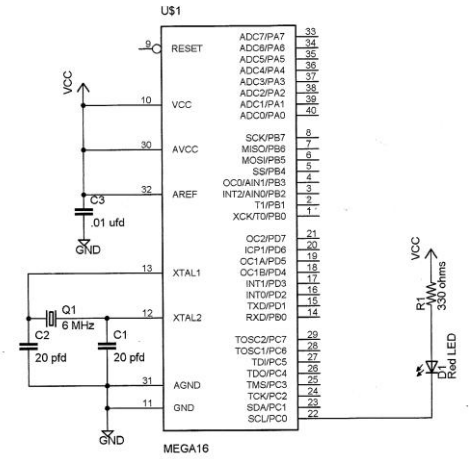
Οι μονάδες των χρονιστών μπορούν να χρησιμοποιούν μεγάλη ποικιλία εσωτερικών συχνοτήτων ,οι οποίες προκύπτουν από την διαίρεση του κεντρικού ρολογιού του μικροελεγκτή ή από μια εξωτερική πηγή. Ο καταχωρητής **timer counter control register** (TCCRx) εμπεριέχει τα bits ελέγχου του προδιαιρέτη CSx2 CSx1 και CSx0 τα οποία ορίζουν την συχνότητα λειτουργίας του χρονιστή καθώς και την πηγή του χρονισμού.
παρακάτω βλέπουμε τον καταχωρητή TCCRx καθώς και τον συνδιασμο των bit που καθορίζουν την πηγή χρονισμού.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
					CS02	CS01	CS00

CS02	CSO1	CS0	
0	0	0	Timer 0 απενεργοποιημένος
0	0	1	Συχνότητα συστήματος CK
0	1	0	Συχνότητα συστήματος/8 CK/8
0	1	1	Συχνότητα συστήματος/64 CK/64
1	0	0	Συχνότητα συστήματος/256 CK/256
1	0	1	Συχνότητα συστήματος/1024 CK/1024
1	1	0	Εξωτερικό Pin T0 (στην καθοδο του παλμού)
1	1	1	Εξωτερικό Pin T0 (στην άνοδο του παλμού)

TIMER 0

Ο timer0 είναι τυπικά ένας χρονιστής εύρους 8 bit. Χρησιμοποιείται συχνά για να δίνει σήμα χρονισμού σε μια διακοπή. Ξεκινάει να μετράει από 0 και μόλις φτάσει την τιμή 255 υπερχειλίζει και ξεκινάει από την αρχή. Την στιγμή της υπερχείλισης προκαλεί μια διακοπή στο πρόγραμμα. Αν στην διακοπή τοποθετήσουμε μια ρουτίνα που θα καταγράφει των αριθμό των διακοπών ,γίνεται κατανοητό πως μπορούμε να πετύχουμε χρόνους μεγάλης ακρίβειας. Έστω λοιπόν πως θέλουμε να κάνουμε ένα led να αναβοσβήνει κάθε 0.5 sec.Το led το τοποθετούμε στο λιγότερο σημαντικό bit της portc ενός μικροελεγκτή mega16. Επειδή θέλουμε η διακοπή να εμφανίζεται κάθε 0.5 sec,είναι προφανές πως θέλουμε ο μετρητής να μετράει για διάστημα 0.5 sec. Όπως βλέπουμε και στο σχήμα παρακάτω η συχνότητα του ρολογιού είναι 6 MHz. Επιλέγοντας προδιαιρέτη χρόνου ισο με 1024 θα έχουμε : 6MHz /1024 =5859KHz όπου η περίοδος θα είναι 1/5859=171μs.Αυτό σημαίνει πως κάθε 171μs θα εμφανίζεται ένας παλμός χρονισμού στον timer0.Επειδή ο timer0 έχει εύρος 8 bit ,μέχρι την υπερχείλιση του θα έχει μετρήσει 255 παλμούς. Άρα 256*171μs=43,69ms. Καταλαβαίνουμε πως ο μέγιστος χρόνος που μπορούμε να μετρήσουμε πριν υπερχειλίσει ο timer0 είναι 43,69 ms. Για να επιτύχουμε χρόνο ίσο με 500ms θα φορτώσουμε στον timer 0 την τιμή 256-59=197(γιατί 59*171μs=10,089ms.Έτσι θα πετύχουμε χρόνο περίπου ίσο με ίσο με 10ms μέχρι να υπερχείλιση ο timer0.Στη συνέχεια θα βάλουμε έναν μετρητή μέσα στην ρουτίνα της διακοπής έτσι ώστε σε κάθε διακοπή να αυξάνει την τιμή του κατά ένα μέχρι να φτάσει τον αριθμό 50.Άρα θα είναι τελικά 50*10ms=500ms=0,5s.Παρακάτω βλέπουμε το κύκλωμα της εφαρμογής μας. καθώς και το ανάλογο software.



```
#include <mega16.h>

static unsigned int time_count;

//ρουτίνα υπερχείλισης του timer0

interrupt [TIM0_0VF] void timer0_ovf_isr(void)

{

        TCNT0=197;          //ορισμός για μέτρηση 10ms

        ++time_count        //αύξησε τον μετρητή

        if (time_count==2950)

         {

                PORTC.0=(PORTC.0^1)     //κάνε XOR το pin 0 της πόρτας c

                                        //για να αναβοσβήσει το led

                time_count=0;

        }

}

void main (void)

{

        DDRC=0x01;                  //κάνε το bit0 της Portc έξοδο

        //θέσε τον προδιαιρέτη για CLK/1024

        TCCR0=0x5;

        TCNT0==0x00;

        OCR0=0x00;

        TIMSK=0x01;                 //ενεργοποίηση διακοπής από τον timer0
```

```
        #asm("sei")                    //ενεργοποίηση διακοπών

        while(1);

}
```

TIMER 1

Ο timer1 είναι πιο πολύπλοκος και χρησιμοποιείται σε περισσότερες εφαρμογές από τον timer 0. Εκτός από τις λειτουργίες του timer0, ο timer1 διαθέτει τον καταχωρητή capture με εύρος 16 bit καθώς επίσης περιέχει και δύο καταχωρητές σύγκρισης εύρους 16 bit. Ο καταχωρητής capture μπορεί να χρησιμοποιηθεί για μέτρηση εύρους παλμού ή να μετράει χρόνους. Οι καταχωρητές σύγκρισης χρησιμοποιούνται για να παράγουν συχνότητες ή παλμούς από τον timer/counter σε ένα pin εξόδου του μικροελεγκτή. Στην συνέχεια θα δούμε αυτές τις δύο λειτουργίες του timer1

TIMER1 PRESCALER

Παρά τις διαφορές που παρουσιάζει ο timer1 σε σχέση με τον timer0 ,ουσιαστικά πρόκειται για έναν δυαδικό μετρητή. Έτσι όπως στον timer0 αλλά και σε όλες τις περιφερειακές μονάδες του μικροελεγκτή υπάρχει ένας καταχωρητής ελέγχου. Αυτός ο καταχωρητής είναι ο TCCR1 (Timer Counter Control Register).Αποτελείται από δύο επιμέρους καταχωρητές τον TCCR1A και τον TCCR1B. Ο TCCR1A ελέγχει την λειτουργία του συγκριτή του timer1 και την λειτουργία της διαμόρφωσης παλμών. Αρχικά όμως θα εξετάσουμε τον TCCR1B. Ο TCCR1B ελέγχει ελέγχει τον προδιαιρέτη, τον πολυπλέκτη εισόδου καθώς και την λειτουργία της σύλληψης (capture). Παρακάτω βλέπουμε τον καταχωρητή TCCR1B

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ICNC1	ICNS1		WGM	WGM	CS12	CS11	CS10

Bit	Λειτουργία
ICNC1	Απποριψη κοινού θορύβου στην λειτουργία της σύλληψης
ICES1	Επιλογή κατωφλιού για σύλληψη (1 κατά την ανοδο ,0 στην καθοδο)
WGM1	Έλεγχος
CS12	Bit επιλογής σήματος χρονισμου. Ακριβώς όπως ο timer0
CS11	Bit επιλογής σήματος χρονισμου. Ακριβώς όπως ο timer0
CS10	Bit επιλογής σήματος χρονισμου. Ακριβώς όπως ο timer0

Η ΛΕΙΤΟΥΡΓΙΑ ΣΥΛΛΗΨΗΣ ΤΟΥ TIMER1(TIMER1 CAPTURE)

Για να μετρήσουμε μια περίοδο χρόνου με τον timer0 υπήρχε η απαίτηση να ξεκινήσουμε τον μετρητή στην αρχή του γεγονότος και να τον σταματήσουμε καθώς τελειώσει αυτό. Στην συνέχεια έπρεπε να διαβάσουμε τον χρόνο από τον καταχωρητή του timer counter.Η ίδια δουλειά γίνεται διαφορετικά στον timer1 διότι ο timer1 συνεχώς μετράει. Για να μετρήσουμε ένα γεγονός καταγράφουμε τον χρόνου της έναρξης του γεγονότος, τον χρόνο λήξης και στην συνέχεια αφαιρούμε τους δύο χρόνους. Για παράδειγμα αν ξεκινήσουμε από την Θεσσαλονίκη για να πάμε στα Τρίκαλα μπορούμε να κοιτάξουμε στο ρολόι μας την ώρα που ξεκινήσαμε από Θεσσαλονίκη και την ώρα που φτάσαμε Τρίκαλα. Έτσι πολύ απλά να αφαιρέσουμε τους χρόνους και να δούμε πόσο χρονικό διάστημα ταξιδεύαμε. Στον timer1 αυτή η διαδικασία ελέγχεται από τον καταχωρητή ICR1 (Input Capure Register)

Ο ICR1 είναι ένας καταχωρητής εύρους 16 bit (εμπεριέχει τους καταχωρητής ICR1H και ICR1L).Το σήμα που θα συλληφθεί μπορεί δώσει εκκίνηση στον ICR1L κατά το κατώφλι ανόδου του ή κατά το κατώφλι καθόδου του. Αυτό ελέγχεται από τον bit ICES1 του TCCR1B.

Επειδή μόνο ένας καταχωρητής υπάρχει διαθέσιμος για την λειτουργία της σύλληψης, γίνεται αντιληπτό πως πρέπει η ανάγνωση του καταχωρητή να γίνει γρήγορα πριν έρθει η επόμενη εγγραφή και μας καταστρέψει τα δεδομένα. Για να μπορεί να είναι εφικτό αυτό μια διακοπή ενεργοποιείται κάθε φορά που νέα δεδομένα συλλαμβάνονται από τον ICR1. Ετσι υπάρχει χρόνος ώστε να σωθούν τα δεδομένα γρήγορα.

Ο timer1 παρέχει επίσης την λειτουργία της απόρριψης κοινού θορύβου στην είσοδο του ICP (input capture pin). Έτσι τυχών απότομες διακυμάνσεις τάσης απορρίπτονται και προστατεύεται έτσι η λειτουργία της σύλληψης από λανθασμένο χρόνο έναρξης. Όταν η λειτουργία απόρριψης κοινού θορύβου είναι ενεργοποιημένη ,τότε το pin ICP πρέπει να κάνει μετρήσει τέσσερα δείγματα πριν αρχίσει να κάνει capture ο μικροελεγκτής. Στο παρακάτω παράδειγμα θα μετρήσουμε την περίοδο ενός τετραγωνικού σήματος κάνοντας χρήση την λειτουργία σύλληψης (capture) με τον μικροελεγκτή atmega16.

```
#include <mega16.h>

//ορισμός της portc ως έξοδο

#define period_out PORTC

unsigned char ov_counter      //μετρητής για την υπερχείλιση του timer1

unsigned int starting_edge,ending_edge

unsigned int clocks;          //θέση αποθήκευσης για τους παλμούς

//ρουτίνα διακοπής του timer1

interrupt [TIM1_OVF] void timer1_ovf_isr(void)

        {

        ++ov_counter ;        //αύξησε τον μετρητή όταν γίνει η διακοπή

        }

//ρουτίνα διακοπής για την λειτουργία της σύλληψης

interrupt [TIM1_CAPT] void timer_capt_isr(void)
```

```
        {
                //συνδυασμός των δύο 8 bit καταχωρητών σε έναν 16 bit
                ending_edge=256*ICR1H+ICR1L;
                clocks=(unsigned long)ending_edge+((unsigned long)ov_counter*65536)
                        -(unsigned long)starting_edge;
                period_out=~(clocks/750);
                ov_counter=0;
                starting_edge=ending_edge;
        }
void main(void)
{
        DDRC=0Xff;  //ορισμός της portc ως έξοδος
        TCCR1A=0;   //απενεργοποίηση όλων των λειτουργιών κυμματομορφών του
                    //timer1
        TCCR1B=0xC2;        //ορισμός για clock/8 και ενεργοποίηση του capture
        TIMSK=0x24;         //ενεργοποίηση διακοπών από υπερχείλιση του timer1 καθώς
                            //και της λειτουργίας capture
        #asm("sei")         //ενεργοποίηση όλων των διακοπών
        while(1)
        {
        ;
        }
}
```

Όταν ξεκινάει να τρέχει το πρόγραμμα όταν ο παλμός που εφαρμόζεται στο pin ICP του μικροελεγκτή βρίσκεται σε άνοδο, ενεργοποιείται η διακοπή capture και ο μετρητής ICR1 αρχίζει να μετράει . Όταν ο timer1 υπερχειλίσει ,τότε ενεργοποιείται το interrupt της υπερχείλισής του. Εκεί αυξάνουμε την τιμή ενός μετρητή κάθε φορά που συμβαίνει η διακοπή. Μέσα στην ρουτίνας διακοπής της σύλληψης (capture) συνδυάζουμε το ICR1H με το ICR1L έτσι ώστε να προκύψει ολόκληρος ο αριθμός των 16 bit. Αυτό μπορούμε να το κάνουμε κάνοντας ολίσθηση τον ICR1H κατά 8 θέσεις δεξιά ή πολύ απλά πολλαπλασιάζοντας τον με τον αριθμό 256. Έπειτα στην μεταβλητή clocks βρίσκουμε πόσους παλμούς χρειάστηκε ο μικροελεγκτής για να καθορίσει την αρχή και το τέλος του προς μέτρηση παλμού. Στην συνέχεια διαιρούμε τον αριθμό των παλμών με την συχνότητα λειτουργίας του timer1 (στην περίπτωσή μας 750Khz) για να βρούμε σε ms πόσο χρόνο ήταν ο προς μέτρηση παλμός.

ΛΕΙΤΟΥΡΓΙΑ ΕΞΟΔΟΥ ΣΥΓΚΡΙΣΗΣ ΤΟΥ TIMER1 (OUTPUT COMPARE)

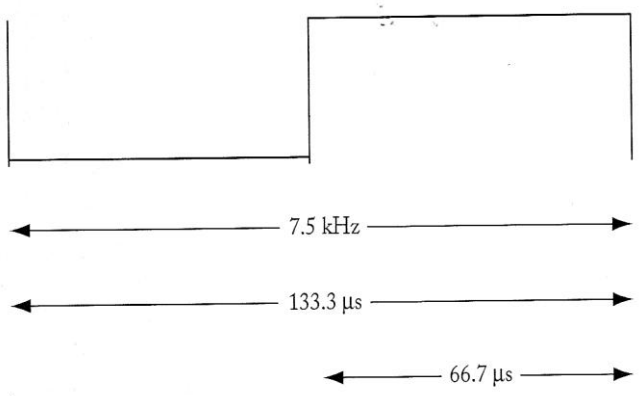
Η λειτουργία εξόδου σύγκρισης χρησιμοποιείται από τον μικροελεγκτή για να παράγει σήματα εξόδου. Τα σήματα αυτά μπορεί να είναι είτε τετραγωνικοί παλμοί ,είτε ασύμμετρα σήματα που μπορεί να ποικίλουν σε συχνότητα ή την συμμετρία τους. Η σύγκριση αποτελεί μια αποκλειστικά εσωτερική διαδικασία στον μικροελεγκτή σε αντίθεση με την λειτουργία σύλληψης όπου έπρεπε να έχει ο μικροελεγκτής εξωτερικό ερέθισμα στο pin ICP. Στην λειτουργία της σύγκρισης το πρόγραμμα φορτώνει μια τιμή στον καταχωρητή σύγκρισης εξόδου. Αυτή η τιμή συγκρίνεται με την τιμή του καταχωρητή timer/counter και αν συμπίπτουν, τότε λαμβάνει χώρα μια διακοπή η οποία δρα σαν συναγερμός προτρέποντας τον επεξεργαστή να εκτελέσει μια συνάρτηση που αντιστοιχεί στο σήμα που την προκάλεσε. Στην συνέχεια βλέπουμε τον καταχωρητή TCCR1A

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Επιπρόσθετα η λειτουργία της σύγκρισης μπορεί να αλλάξει την λογική κατάσταση του προκαθορισμένο pin OC1x ,σύμφωνα με τον παρακάτω πίνακα.

COM1x1	COM1x0	Αποτέλεσμα
0	0	Χωρίς έξοδο
0	1	Εναλλαγή λογικού επιπέδου στο OC1x
1	0	Τοποθέτηση σε λογικό 0 του OC1x
1	1	Τοποθέτηση σε λογικό 1 του OC1x

Για παράδειγμα έστω πως θέλουμε ο μικροελεγκτής να παράγει ένα τετραγωνικό σήμα συχνότητας 7,5Khz.Δηλαδή η περίοδός του είναι 1/7500Hz=133μς. Αν η συχνότητα του ρολογιού μας είναι 6Mhz, αυτό σημαίνει πως με έναν προδιαιρέτη CLK/8 θα έχουμε συχνότητα 750Khz(η περίοδος 1,33μς). Εμείς θέλουμε ο παλμός μας να εναλλάσσεται κάθε 133μς/2=66,7μς.Άρα έχοντας ως δεδομένο την συχνότητα του ρολογιού πρέπει να ισχύει για την αλλαγή της κατάστασης του pin OC1A 66,7μς/1.33μς=50 κύκλοι ρολογιού όπως βλέπουμε στο σχήμα

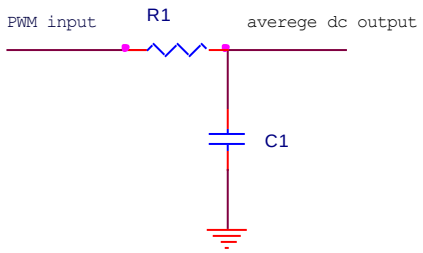


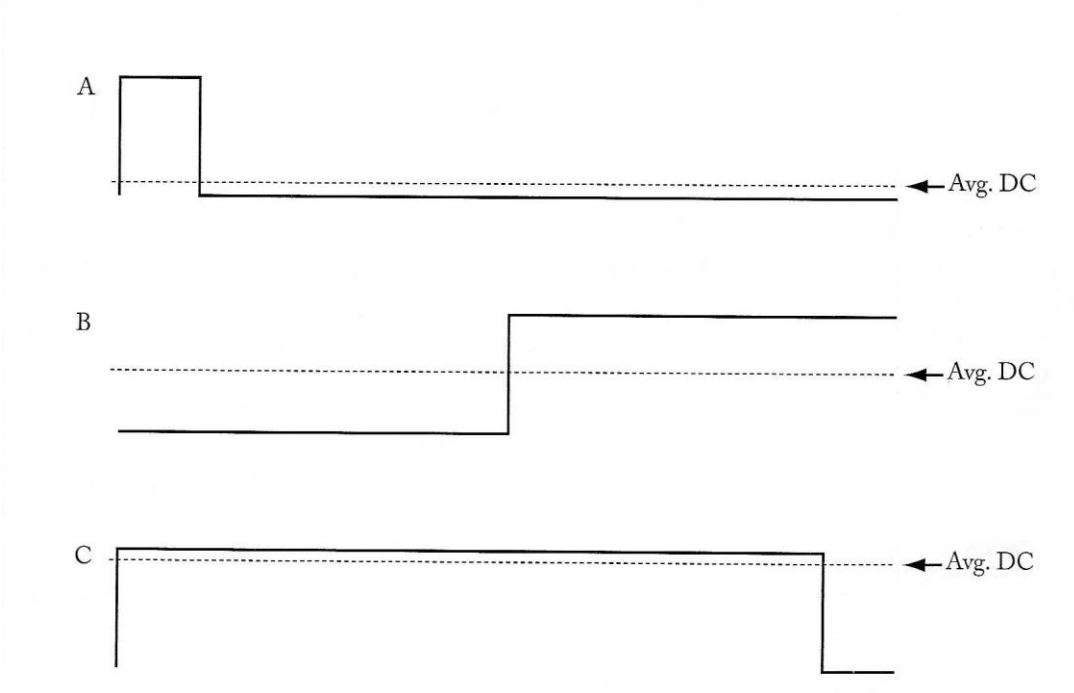

```
#include <mega16.h>

interrupt [TIM1_COMPA] void timer1_compa_isr(void)

        {
```

```
        OCR1A=OCR1A+50;         //εναλλαγή εξόδου σε 50 κύκλους
                                //ο OCR1A εμπεριέχει τον OCR1AH και τον
                        //OCR1AL
        }
void main(void)
{
        DDRD=0x20;              //ρύθμιση του pin PD5 ως έξοδο
        TCCR1B=0x40;            //ρύθμιση για εναλλαγή κάθε φορά που έχουμε ταίριασμα
        TCCR1B=0x02;            //ρύθμιση του ρολογιού για CLK/8
        TIMSK=0x20;             //ενεργοποίηση των διακοπών λόγω σύγκρισης
        #asm ("sei")            //ενεργοποίηση όλων των διακοπών
}
while(1);
}
```

ΔΙΑΜΟΡΦΩΣΗ ΠΛΑΤΟΥΣ ΠΑΛΜΩΝ ΜΕ ΤΟΝ TIMER1 (PWM)

Μια από τις μεθόδους για να μετατρέψουμε ένα ψηφιακό σήμα σε αναλογικό(digital to analog converter), είναι με την χρήση διαμόρφωσης πλάτους παλμών (PWM). Μεταβαλλόντας το duty cycle ενός παλμού η μέση τιμή της τάσης μεταβάλλεται αντίστοιχα. Φιλτράροντας την έξοδο του παλμού, μπορούμε να πάρουμε την επιθυμητή τιμή της τάσης που μας ενδιαφέρει.Το φιλτράρισμα μπορεί να γίνει με ένα απλό δικτύωμα RC όπως βλέπουμε στην παρακάτω εικόνα





Ο timer1 αλλάζει την λειτουργία του έτσι ώστε να είναι σε θέση να παράγει παλμούς PWM με την ενσωματωμένη αυτή λειτουργία του. Κατά την διάρκεια της λειτουργίας PWM ο timer1 ξεκινάει να μετράει από το μηδέν μέχρι μια άνω τιμή που του ορίζουμε και από την άνω τιμή ως το μηδέν. Η άνω τιμή εξαρτάται από την ανάλυση που του ορίζουμε. Έχουμε την δυνατότητα να του ορίσουμε ανάλυση των 8 bit των 9 bit και των 10 bit αλλάζοντας τον συνδυασμό των WGM bit στον καταχωρητή TCCR1A. Ο συνδυασμός αυτός των bit επηρεάζει την τελική τιμή που θα φτάσει ο μετρητής, καθώς και την συχνότητα της PWM κυμματομορφής. Παρακάτω βλέπουμε τα bit WGM11 και WGM10 του καταχωρητή TCCR1A.

WGM11	WGM10	Ανάλυση PWM	Άνω τιμή timer
0	0	Pwm off	Pwm off
0	1	8-bit	255
1	0	9-bit	511
1	1	10-bit	1024

Εάν για παράδειγμα επιλέξουμε το ρολόι του συστήματος να τρέχει με 6Mhz ,προδιαιρέτη CLK/8 και ανάλυση των 8 bit θα έχουμε ως αποτέλεσμα η άνω τιμή του timer1 να είναι 511. Η συχνότητα τότε του pwm παλμού θα μας δίνεται από την σχέση $f_{pwm}=f_{system\ clock}$/(προδιαιρέτη*2*άνω τιμή) Έτσι έχουμε 6Mhz/(8*2*511)= 733.8Hz .Κατά τον σχεδιασμό ενός συστήματος πρέπει να λάβουμε υπόψιν μας την επιθυμητή τιμή συχνότητας που θέλουμε να επιτύχουμε και να δοκιμάσουμε ποια ανάλυση είναι αυτή που πλησιάζει περισσότερο στη ζητούμενη συχνότητα.

Το duty cycle του PWM παλμού εξαρτάται από την τιμή που είναι καταχωρημένη στον καταχωρητή σύγκρισης εξόδου. Αν για παράδειγμα θέλουμε ένα duty cycle 20% θα πρέπει να φορτώσουμε στον output compare register μια τιμή που να είναι 20% της τιμής κορυφής του παλμού pwm. Στο παρακάτω παράδειγμα το πρόγραμμα παράγει τέσσερα διαφορετικά duty cycle(10%,20% 30% και 40%) μιας συχνότητας που είναι κοντά στα 2Khz. Έχοντας έναν κρύσταλλο των 8 Mhz και διαλέγοντας ανάλυση των 8-bit με προδιαιρέτη CLK/8 ,σύμφωνα με τον παραπάνω τύπο παίρνουμε μια συχνότητα 1,96 Khz η οποία είναι πολύ κοντά στην ζητούμενη συχνότητα.

```
#include <mega16.h>

#define PWM_select (PINA & 3)

void main(void)

{

        unsigned int old_sw ;          //αποθήκευση παλαιών τιμών των διακοπτών
```

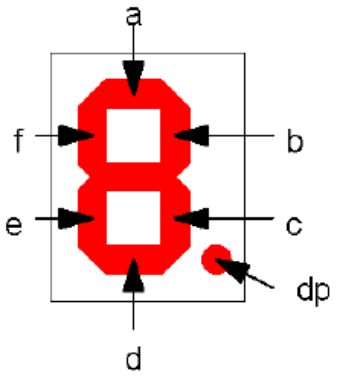
```
PORTA=0x03;                    //κάνε τα 2 lsb της porta είσοδο
DDRD=0x20;                     //ορισμός του OCA ως έξοδο
TCCR1A=0x9          1//σύγκριση για χαμηλη εξοδο ,ενεργοποίηση του Pwm
                                        //και ανάλυση 8-bit
TCCR1B=0x02;                   //πηγή χρονισμού το ρολόι και ο προδιαιρέτης CLK/8


while(1)
{
        if (PWM_select !=old_sw)
        {
        old_sw=PWM_select
        switch (PWM_select)
                {
                case 0: OCR1A=25;          //10 % duty cycle
                        break;
                case 1: OCR1A=51;          //20 % duty cycle
                        break;
                case 2:OCR1A=76;           //30 % duty cycle
                        break;
                case 3:OCR1A=102;          // 40 % duty cycle
                        break;
                default:
                }
```

```
            }
        }
}
```

Στο παραπάνω πρόγραμμα χρησιμοποιήσαμε την συνθήκη switch case η οποία ανάλογα σε ποια κατάσταση βρίσκεται η μεταβλητή PWM_select εκτελεί το αντίστοιχο case. Το πρόγραμμα αυτό θα εκτελείται συνεχώς, καθώς η συνθήκη while(1) ισχύει πάντα.

ΑΠΕΙΚΟΝΙΣΗ ΣΕ ΕΝΔΕΙΚΤΗ LED 7 ΤΟΜΕΩΝ

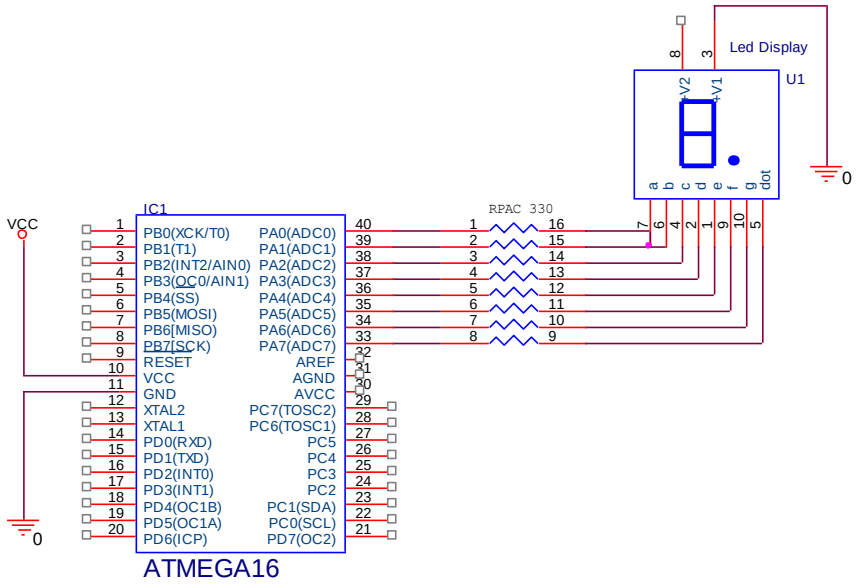
Ο ενδείκτης 7 τομέων αποτελείται από τους τομείς (a, b, c, d, e, f, g) κατάλληλα τοποθετημένους έτσι ώστε να είναι δυνατή η απεικόνιση αριθμών και κατ' επέκταση ορισμένων γραμμάτων. Κάθε τομέας του ενδείκτη αποτελείται από ένα ή περισσότερα leds ανάλογα με το μέγεθος του. Πέρα από τους 7 τομείς στους ενδείκτες υπάρχει τις περισσότερες φορές και ένας επιπλέον τομέας γνωστός ως δεκαδική τελεία (decimal point ή dp). Χρησιμεύει στις περιπτώσεις όπου απαιτείται η ένδειξη δεκαδικών ψηφίων σε απεικόνιση με 2 ή περισσότερους ενδείκτες.



Οι ενδείκτες διακρίνονται σε δύο είδη: σε ενδείκτες κοινής καθόδου οι οποίοι συμβαδίζουν με τη θετική λογική (έχουν κοινή την κάθοδο όλων των leds) και σε ενδείκτες κοινής ανόδου οι οποίοι συμβαδίζουν με τη αρνητική λογική (έχουν κοινή την άνοδο όλων των leds). Στο παραπάνω σχήμα βλέπουμε ποιους τομείς πρέπει να ανάψουμε ανάλογα με το ψηφίο που θέλουμε να εμφανίσουμε. Στην συνέχεια θα οδηγήσουμε έναν ενδείκτη 7 τομέων κοινής καθόδου με τον μικροελεγκτή ATMEGA16.Το πρόγραμμα που θα δούμε

εμφανίζει στο led display τους αριθμούς από το 0 μέχρι το 9 με χρονική διαφορά μεταξύ τους ενός δευτερολέπτου. Μόλις ο αριθμός φτάσει στο 9, τότε αρχίζει πάλι από το 0.

```
#include <mega16.h>
#include <delay.h>
int i;
unsigned number[10]={                   //αποθηκεύουμε τους τομείς που θέλουμε να ανάβουμε
0b00111111          //0                 //κάθε φορά στον πίνακα number
0b00000011          //1
0b01011011          //2
0b01001111          //3
0b01100110          //4
0b01101101          //5
0b01111100          //6
0b00000111          //7
0b01111111          //8
0b01100111          //9
        };
         main()
{
        DDRA=0xFF;                      //H PORTA ως έξοδος

        while(1)
        {
           PORTA=number[i];             //διάβασε τα περιεχόμενα του πίνακα στην θέση i
          delay_ms(1000);               //και οδήγησέ τα στην porta.
           i=i+1;
           if (i>9)
          i=0;
        }
}
```

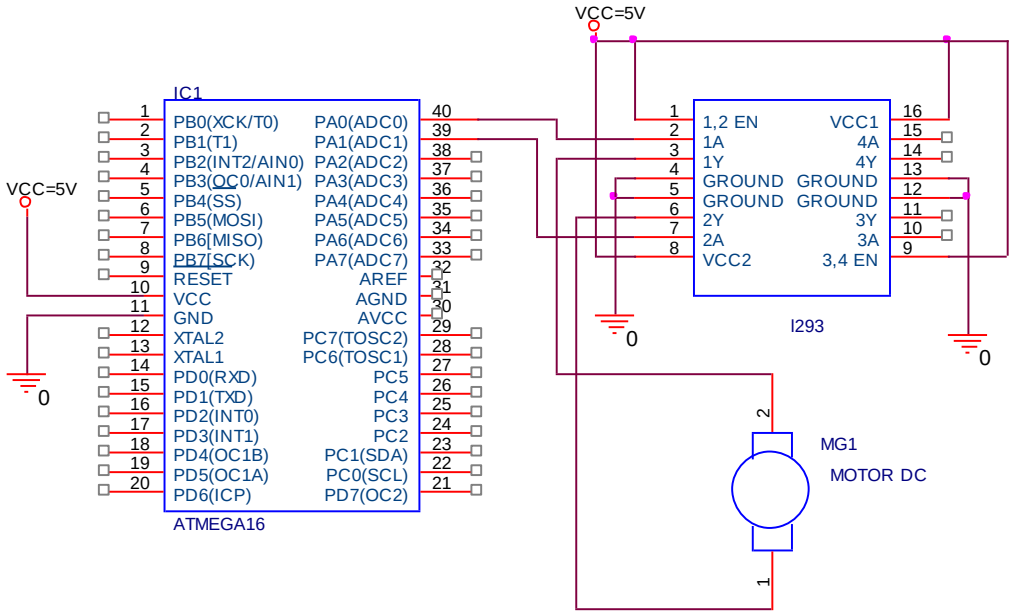


ΟΔΗΓΗΣΗ ΚΙΝΗΤΗΡΩΝ ΜΕ ΤΟΥΣ ΜΙΚΡΟΕΛΕΓΚΤΕΣ AVR

Οι κινητήρες είναι εξαρτήματα με αυξημένες ανάγκες για ρεύμα. Οδηγώντας έναν κινητήρα απευθείας από την έξοδο του μικροελεγκτή θα οδηγούσαμε στην καταστροφή του γιατί έχουν ως ανώτατο όριο ρεύματος εξόδου τα 40mA. Υπάρχουν αρκετοί τρόποι οδήγησης μεγάλων φορτίων από μικροελεγκτές. Στα παρακάτω παραδείγματα έχει χρησιμοποιηθεί μια γέφυρα Η σε μορφή ολοκληρωμένου κυκλώματος. Στην συνέχεια θα δούμε πως μπορούμε να οδηγήσουμε βηματικούς κινητήρες (μονοπολικούς και διπολικούς), κινητήρες servo, αλλά και απλούς DC κινητήρες.

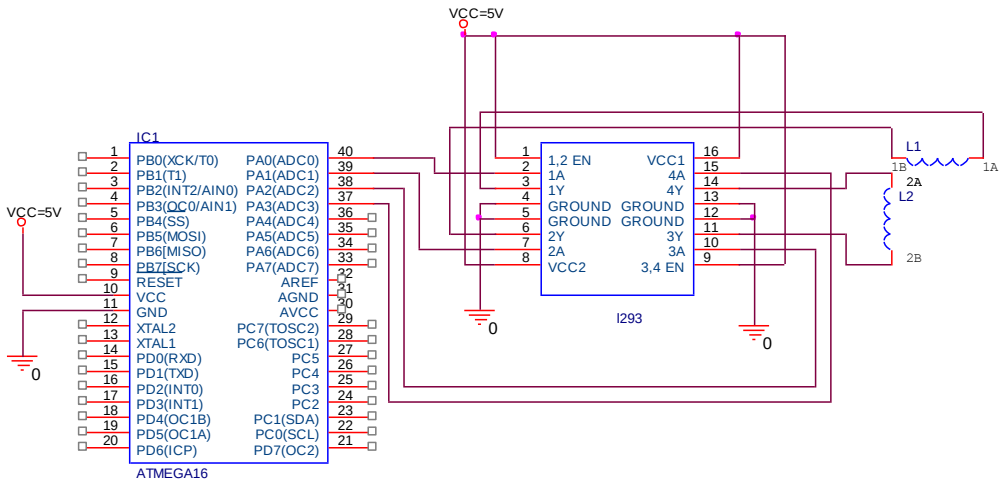
ΟΔΗΓΗΣΗ ΚΙΝΗΤΗΡΩΝ DC

Στο παρακάτω παράδειγμα θα οδηγήσουμε έναν κινητήρα dc μικρής ισχύος με χρήση του ολοκληρωμένου κυκλώματος L293. Το συγκεκριμένο ολοκληρωμένο μπορεί να οδηγήσει 2 διαφορετικούς dc κινητήρες. Το ρεύμα που μπορεί να δώσει είναι 1A ανά κανάλι (ή ανά κινητήρα). Η τροφοδοσία του είναι διπλή και μπορεί να είναι από 4,5V μέχρι 36V. Σε περίπτωση που οι κινητήρες μας "τραβάνε" περισσότερο από 300 mA ρεύμα, είναι προτιμότερο να τοποθετήσουμε μία ψήκτρα πάνω στο ολοκληρωμένο, για να μην έχουμε θερμική φυγή και καταστροφή του. Παρακάτω βλέπουμε στο σχηματικό έναν μικροελεγκτή ATMEGA16 να οδηγεί έναν κινητήρα συνεχούς ρεύματος.



ΟΔΗΓΗΣΗ ΒΗΜΑΤΙΚΟΥ ΚΙΝΗΤΗΡΑ

Οι βηματικοί κινητήρες, είναι μηχανές που μετατρέπουν ακολουθίες ηλεκτρικών παλμών σε μηχανική κίνηση. Χρησιμοποιούνται σε πάρα πολλές σύγχρονες οικιακές και επαγγελματικές συσκευές όπως στους εκτυπωτές και στα φωτοτυπικά μηχανήματα, έχοντας ως καθήκοντα την ακριβή περιστροφή των αξόνων τους . Ένας βηματικός κινητήρας χαρακτηρίζεται από ένα πλήθος παραμέτρων όπως η τιμή τους, το μέγεθος, το σχήμα,η ισχύς,η τάση τροφοδοσίας εκ των οποίων η σημαντικότερη αφορά το πλήθος των βημάτων που απαιτούνται για μια πλήρη περιστροφή του ρότορα. Η περιστροφή του ρότορα μπορεί να γίνεται με πλήρη η με μισά βήματα. Η οδήγηση με μισά βήματα αυξάνει πάντα την ακρίβεια μετατόπισης του ρότορα, γεγονός που έχει σαν συνέπεια την ακόμα πιο ακριβή περιστροφή του ελεγχόμενου μηχανικού στελέχους.



Παραπάνω βλέπουμε τα 2 πηνία του βηματικού κινητήρα συνδεμένα στην έξοδο του L293. Τα σήματα ελέγχου, όπως βλέπουμε στον παρακάτω πίνακα, πρέπει να ανταποκρίνονται σε μια συγκεκριμένη ακολουθία τεσσάρων βημάτων έτσι ώστε να έχουμε στρέψη του κινητήρα.

	Τυλιγμα 1A	Τυλιγμα 2A	Τυλιγμα 1B	Τυλιγμα 2B
ΒΗΜΑ 1	1	1	0	0
ΒΗΜΑ 2	0	1	1	0
ΒΗΜΑ 3	0	0	1	1
ΒΗΜΑ 4	1	0	0	1

Στην συνέχεια βλέπουμε το αντίστοιχο πρόγραμμα στον μικροελεγκτή μας.

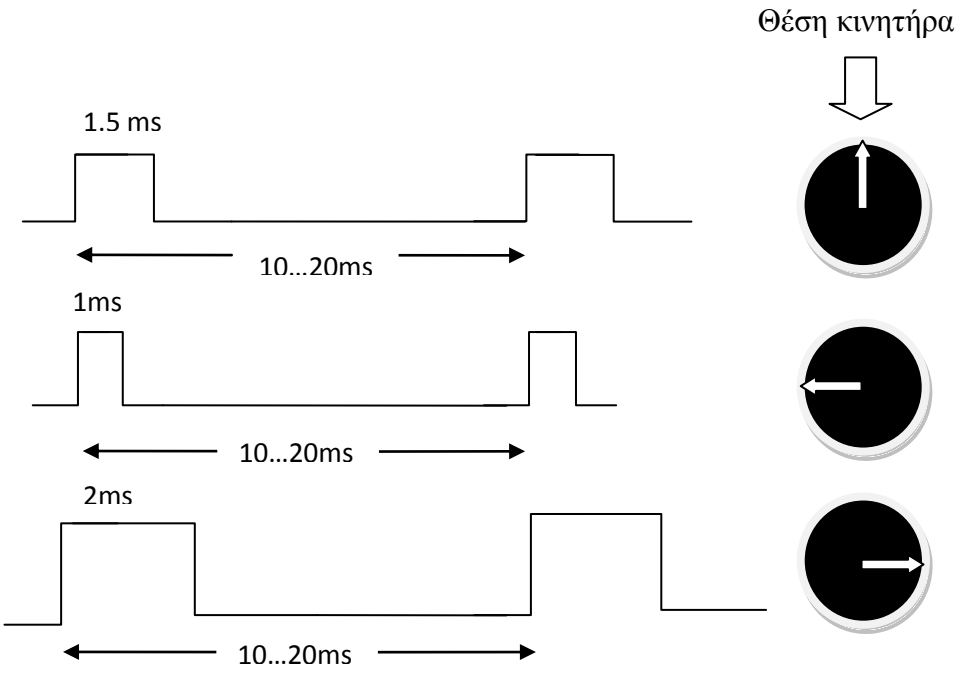
```
#Include<mega16.h>
#include<delay.h>
main()
{
DDRA=0xFF;          //ορισμός της porta ως έξοδο
while(1)
        {
     PORTA=0b00001100;
    delay_ms(20);
    PORTA=0b00000110;
    delay_ms(20);
    PORTA=0b00000011;
```

```
    delay_ms(20);
    PORTA=0b00001001;
    delay_ms(20);
      }
}
```

ΟΔΗΓΗΣΗ ΚΙΝΗΤΗΡΑ SERVO

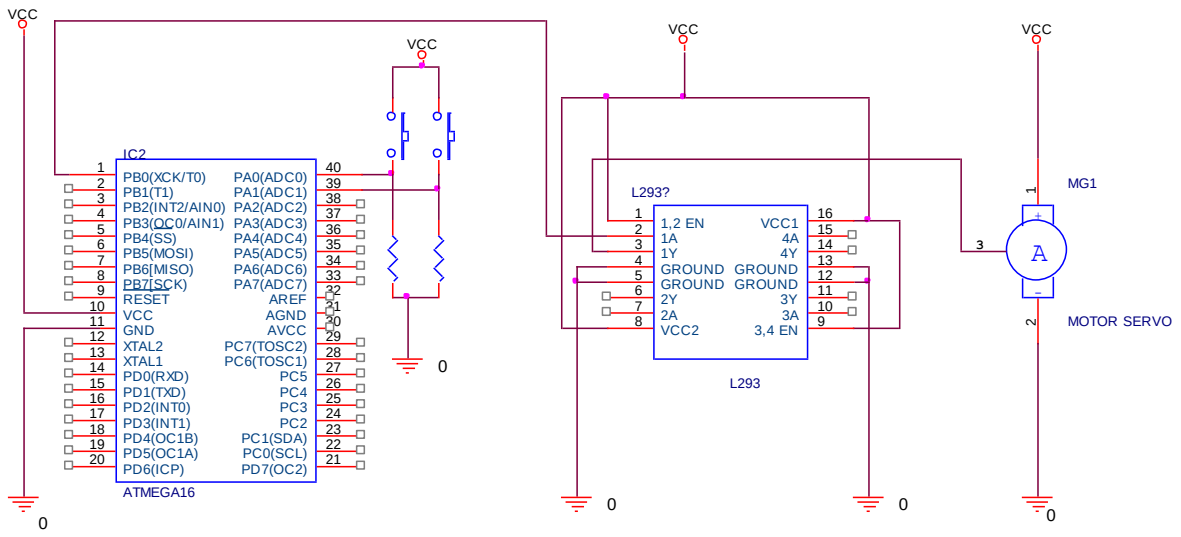
Οι σημερινοί σερβοκινητήρες είναι εξαιρετικά συμπαγείς, παρότι εκτός από τον ίδιο τον μηχανισμό κίνησης περιλαμβάνουν εξειδικευμένα ηλεκτρονικά οδήγησης η είσοδος των οποίων χρησιμοποιεί απλά λογικά σήματα TTL η CMOS. Η ισχύς την οποία είναι σε θέση να παράσχουν οι ισχυρότεροι εξ αυτών μπορεί να είναι αρκετά μεγάλη, ενώ συχνά συνοδεύονται από ένα σεβαστό πλήθος αξεσουάρ όπως βραχίονες, και διάτρητοι τροχοί τα οποία διευκολύνουν την προσαρμογή τους με τα κινούμενα στελέχη.

Ένας σερβοκινητήρας διαθέτει μόλις 3 καλώδια χρωματικά κωδικοποιημένα. Το κόκκινο και το μαύρο καλώδιο χρησιμοποιούνται για την τροφοδοσία του κινητήρα, με μια τάση μεταξύ 4,8V και 6V. Το τρίτο καλώδιο (συνήθως κίτρινο η λευκό), χρησιμοποιείται για να μεταφέρει στον σερβοκινητήρα τις εντολές με την μορφή διαμορφωμένων παλμών κατά πλάτος (PWM) σημάτων. Ο ρυθμός επανάληψης θα πρέπει να είναι τέτοιος ώστε μεταξύ των διαδοχικών παλμών η παύση να μην είναι μεγαλύτερη από 10ms εώς 20ms. Ο ρυθμός επανάληψης δεν είναι κρίσιμος, αλλά βοηθά τον κινητήρα να διατηρήσει την θέση που ορίζει το πλάτος των λαμβανόμενων παλμών. Εάν οι παλμοί δεν επαναλαμβάνονται, ο σερβοκινητήρας θα οδηγηθεί στην θέση που υποδεικνύει ο τελευταίος ληφθείς παλμός, αλλά μόλις αυτός σταματήσει η ελάχιστη δύναμη στον άξονα μπορεί να μετακινήσει τον κινητήρα και να χάσει την επιθυμητή θέση.



Ένας παλμός διάρκειας 1.5ms φέρνει τον σερβοκινητήρα στην θέση ηρεμίας, τις 90 μοίρες. Ένας παλμός διάρκειας 1ms υποχρεώνει τον κινητήρα να περιστραφεί προς την ακραία αριστερή θέση του που στην συγκεκριμένη περίπτωση είναι 0 μοίρες. Αντίστοιχα σε έναν παλμό διάρκειας 2 ms μπορούμε να γυρίσουμε τον κινητήρα ως τις 180 μοίρες. Ενδιάμεσες τιμές είναι δυνατόν να επιτευχθούν μεταβάλλοντας το πλάτος του παλμού μεταξύ 1ms και 2 ms. Στην συνέχεια βλέπουμε τον κώδικα όπου μπορούμε να ελέγξουμε την φορά κίνησης ενός σερβοκινητήρα.

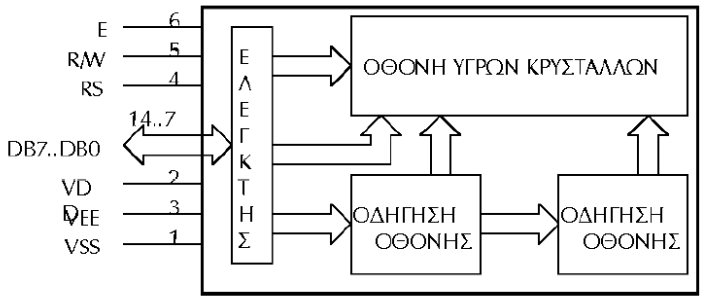
```
#include   <mega16.h>
#include <delay.h>
main()
{
DDRB=0xFF;          //Ορισμός της portb ως έξοδο
DDRA=0x00;          //Ορισμός της Porta ως είσοδο.
 int i;
i=0;
While(1)
{
        If (PORTA.0==1)

        {
                i=i+50; //αν το Porta.0 είναι 1 τότε μετατόπισε αριστερά
        }
        If (PORTA.1==1)
        {
                i=i-50; αν το Porta.1 είναι 1 τότε μετατόπισε δεξιά
        }
PORTB.0==1;
delay_us(1500-i);
PORTB.0==0;
delay_ms(20);
                }
        }
```



Το κύκλωμα οδήγησης του σερβοκινητήρα.

ΟΔΗΓΗΣΗ ΟΘΟΝΗΣ ΥΓΡΩΝ ΚΡΥΣΤΑΛΩΝ

Πολλές από τις σύγχρονες ηλεκτρονικές εφαρμογές απαιτούν μονάδα απεικόνισης με περισσότερες δυνατότητες από τους ενδείκτες των 7 τομέων. Η εταιρία Hitachi έχει δημιουργήσει έναν ελεγκτή με όνομα HD44780 ο οποίος έχει καθιερωθεί και είναι σε θέση να ελέγχει την απεικόνιση σε οθόνες υγρών κρυστάλλων. Πολλές εταιρίες κατασκευάζουν μονάδες απεικόνισης χρησιμοποιώντας τον ελεγκτή της εταιρίας Hitachi ή και ελεγκτές τρίτων εταιριών. Μια μονάδα οθόνης υγρών κρυστάλλων αποτελείται από μία οθόνη, έναν ελεγκτή και δύο ολοκληρωμένα οδήγησης της οθόνης όπως βλέπουμε παρακάτω.

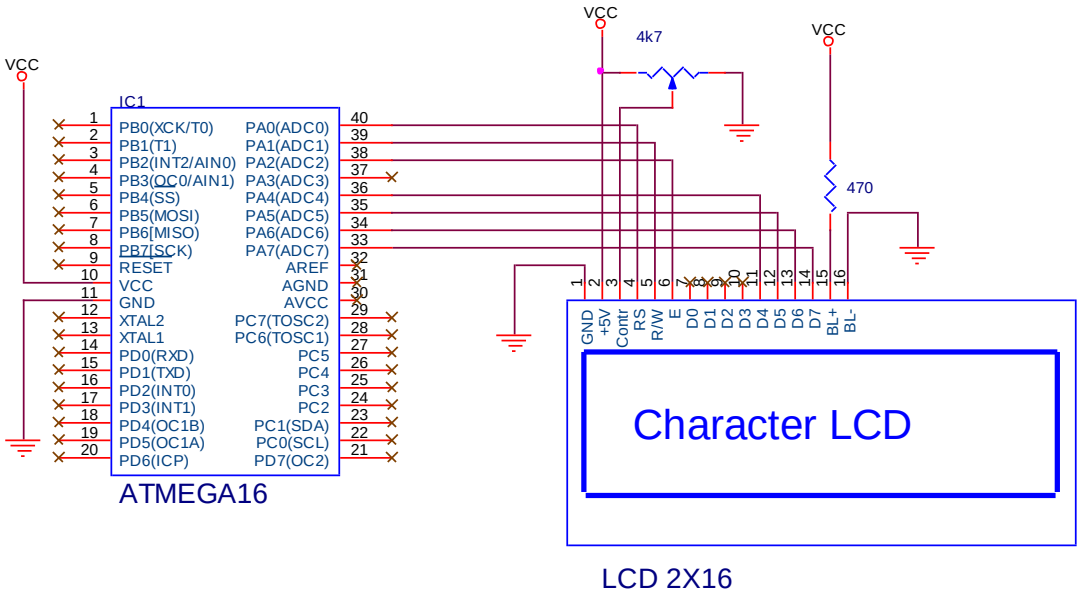


Σε μία οθόνη υγρών κρυστάλλων διακρίνουμε τα pins που βλέπουμε στο παραπάνω σχεδιάγραμμα.

PIN	ΠΕΡΙΓΡΑΦΗ
E	ENABLEΑρχικοποίηση μετάδοσης. Εγγραφή δεδομένων κατά την εναλλαγή από high σε low. Αντίστοιχα ανάγνωση δεδομένων κατά την εναλλαγή από low se high.
R/W	READ/WRITE.Όταν είναι 0 μπορούμε να γράψουμε δεδομένα στην οθόνη ,ενώ όταν είναι 1 μπορούμε να διαβάσουμε δεδομένα από αυτήν
RS	REGISTER SELECT. Όταν είναι 0 τα bytes που στέλνονται στην οθόνη λαμβάνονται ως εντολές. Διαφορετικά λαμβάνονται ως δεδομένα χαρακτήρων
DB7…DB0	Διεύθυνση δεδομένων εύρους 8 bit.
VD	Ρύθμιση αντίθεσης της οθόνης
VEE	Τροφοδοσία
VSS	γείωση

Για να στείλουμε δεδομένα στην οθόνη πρέπει να ακολουθήσουμε ορισμένους προκαθορισμένους κανόνες αρχικοποίησης που βρίσκουμε στο datasheet αυτής. Ο μεταγλωττιστής codevision μας προσφέρει έτοιμες τις αρχικοποιήσεις αυτές απλουστεύοντας έτσι την προσπέλαση της δίνοντας μας την δυνατότητα να παράγουμε γρήγορα και εύκολα εφαρμογές που στηρίζονται σε οθόνη υγρών κρυστάλλων. Στην συνέχεια θα εμφανίσουμε σε lcd 2x16 που είναι συνδεμένη στην PORTA ενός μικροελεγκτή ATMEGA16 την λέξη “HELLO”.

```
#Include <mega16.h>
#asm            //περιλαμβάνουμε κώδικα assembly στο πρόγραμμά μας.
.equ __lcd_port=0x1B ;PORTA        //ορισμός της porta ως την πόρτα που έχουμε
#endasm                            // συνδεμένη την lcd οθόνη.
#include <lcd.h>       //περιλαμβάνουμε τις ρουτίνες οδήγησης της οθόνης
main()
{
DDRA=0xFF;             //Η PORTA ως έξοδος
lcd_init(16);          //αρχικοποίηση της οθόνης για 2 γραμμές και 16 στήλες
lcd_gotoxy(0,0);       //εντολή καθορισμού σημείο εκκίνησης εγγραφής στην οθόνη. Το
                       //κείμενο θα ξεκινήσει από την θέση 0,0 της οθόνης
printf("HELLO");       //εμφάνιση του μηνύματος HELLO
}
```



Όπως βλέπουμε στην οθόνη τα D0-D3 δεν χρησιμοποιούνται. Έτσι εξοικονομούμε χώρο στο pcb της εφαρμογή μας. Η οδήγηση με 4 bit δεδομένων δεν απασχολεί τον προγραμματιστή διότι εμπεριέχεται στις ρουτίνες του μεταγλωττιστή.

ΣΕΙΡΙΑΚΗ ΕΠΙΚΟΙΝΩΝΙΑ ΜΕ ΤΟ USART

Η σειριακή επικοινωνία είναι μια διαδικασία αποστολής πολλαπλών bit δεδομένων μέσω ενός απλού καλωδίου. Τα bits μιας σειριακής αποστολής χωρίζονται μεταξύ τους από προκαθορισμένα χρονικά όρια. Το USART χρησιμοποιείται για να επικοινωνήσει ένας μικροελεγκτής με διάφορες άλλες συσκευές. Μια από αυτές οι συσκευές είναι και ο υπολογιστής όπου μέσω της σειριακής θύρας μπορεί να έχει αμφίδρομη επικοινωνία με τον μικροελεγκτή .

Η συνήθης μορφή επικοινωνίας είναι η ασύγχρονη σειριακή επικοινωνία (asynchronous serial communication). Είναι ασύγχρονη με την έννοια πως δεν χρειάζεται κοινό ρολόι η μονάδα αποστολής των δεδομένων με την μονάδα λήψης αυτών. Η ασύγχρονη μετάδοση ξεκινάει με ένα bit εκκίνησης (start bit) και σταματάει με ένα bit περάτωσης λειτουργίας (stop bit).

κατάσταση	Off	off	on	on	on	on	on	on	on	on	on	on	off
Λογικό επίπεδο	1	1	0	1	0	0	0	0	0	1	0	1	1
Τύπος bit			start	data	data	data	data	data	data	data	data	stop	
Αριθμός δεδομένων				0	1	2	3	4	5	6	7		

Στον παραπάνω πίνακα βλέπουμε την αλληλουχία των bit δεδομένων σε σχέση με το λογικό επίπεδο της γραμμής. Αρχικά το start bit δηλώνει την αρχή της μετάδοσης και ακολουθούν τα bit δεδομένων ξεκινώντας από το λιγότερο σημαντικό ψηφίο του byte και καταλήγοντας στο σημαντικότερο ψηφίο. Στη συνέχεια έρχεται το stop bit που δηλώνει το τέλος της μετάδοσης. Όταν το start bit ρίξει το λογικό επίπεδο στο 0, τότε ο δέκτης ξεκινάει να λαμβάνει δεδομένα εφόσον περιμένει για χρονικό διάστημα ίσο με 1.5 φορά τον χρόνο του bit. Στην συνέχεια ο δέκτης αφιερώνει ένα χρόνο bit για κάθε bit έτσι ώστε να εξασφαλίσει την ακρίβεια και την αξιοπιστία στην μετάδοση. Όλη η διαδικασία του χρονισμού δεν απασχολεί άμεσα τον προγραμματιστή.

Το μόνο που θα απασχολήσει τον προγραμματιστή,είναι να σιγουρευτεί πως οι παράμετροι της σειριακής επικοινωνίας του πομπού και του δέκτη είναι ίδιοι. Έτσι πρέπει να δηλώσει τον σωστό αριθμό των bits (συνήθως 8), να δηλώσει πότε επιθυμεί bit ισοτιμίας και να θέσει τον ρυθμό μετάδοσης.

Το USART μπορεί να χρησιμοποιηθεί με πολλά μέσα μετάδοσης όπως ραδιοκύματα, με υπέρυθρες ακτίνες, με laser καθώς και με χάλκινα κανάλια. Ο πιο διαδεδομένος τρόπος είναι το RS-232. Κατασκευάστηκε ειδικά ώστε να είναι σε θέση να μεταφέρει σειριακά σήματα με αξιοπιστία σε σχετικά αρκετά μεγάλες αποστάσεις. Στο RS-232 το λογικό 1 εκφράζεται ανεστραμμένα με μια τάση πιο αρνητική από -3 volt. Αντίστοιχα το λογικό 0 εκφράζεται με μια θετική τάση μεγαλύτερη από +3 volt. Χρησιμοποιώντας λογικές τάσεις διαφορετικές από το 0, γίνεται πιο εύκολος ο έλεγχος για

σφάλματα στο σύστημα, επειδή ένα σπασμένο καλώδιο θα παρουσιάζει τάση 0 volt κάνοντας έτσι την βλάβη ανιχνεύσιμη.

Ο compiler codevision της HP infotech εμπεριέχει βιβλιοθήκες για τον έλεγχο της σειριακής μετάδοσης. Η μετάδοση συνήθως καταλήγει σε μια τερματική συσκευή, όπως για παράδειγμα ένας προσωπικός υπολογιστής που εκτελεί ένα τερματικό πρόγραμμα. Σε αυτό μπορούμε να στείλουμε σειριακά δεδομένα και να τα εμφανίζουμε στην οθόνη. Ο codevision εμπεριέχει ένα τερματικό πρόγραμμα για αυτή ακριβώς την εφαρμογή.

Ένα απλό παράδειγμα θα δούμε παρακάτω, όπου θα κάνουμε χρήση της συνάρτησης sprintf() και θα στείλουμε την λέξη "HELLO " σε ένα τερματικό. Η συνάρτηση sprintf() εμπεριέχεται στο stdio.h το οποίο θα συμπεριλάβουμε στην αρχή του προγράμματος.

```
#include <mega16.h>
#include<stdio.h>              //καθιερωμένες συναρτήσεις εισόδου εξόδου
Void main(void)
{
        UCR=0x18;              //αρχικοποίηση της σειριακής πόρτας
        UBRR=0x33;             //ορισμός ρυθμού μετάδοσης
        printf ("\n\ HELLO");                  //εκτύπωση της λέξης HELLO
        while (1);
}
```

Η διασύνδεση USART αποτελείται από 3 καταχωρητές. Τον καταχωρητή ελέγχου και κατάστασης, τον καταχωρητή ρυθμού μετάδοσης και τον καταχωρητή δεδομένων. Αυτοί οι καταχωρητές μπορούν να έχουν μήκος είτε 16 bit είτε 8 bit, ανάλογα με την πολυπλοκότητα του USART. Στον atmega16 σε κανονική ασύγχρονη λειτουργία, ο καταχωρητής ελέγχου του USART έχει μήκος 16 bit αποτελούμενος από 2 καταχωρητές των 8 bit, τους UCSRA και τον UCSRB. Ο καταχωρητής ρυθμού μετάδοσης αποτελείται επίσης από τον UBRRH και τον UBRRL οι οποίοι σε συνδιασμό αποτελούν τον UBRR. Παρακάτω βλέπουμε τον καταχωρητή ελέγχου του USART τον UCSRB

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

BIT	ΠΕΡΙΓΡΑΦΗ
RXCIE	Ενεργοποίηση διακοπής λόγο λήψης δεδομένων
TXCIE	Ενεργοποίηση διακοπής λόγο αποστολής δεδομένων
UDRIE	Ενεργοποίηση διακοπής λόγο μηδενισμού του καταχωρητή δεδομένων
RXEN	Θέτουμε για ενεργοποίηση σειριακής λήψης
TXEN	Θέτουμε για ενεργοποίηση σειριακής αποστολής
UCSZ2	Θέτει αριθμό bit δεδομένων
RXB8	Το ένατο bit λήφθηκε σε (λειτουργία 9-bit)
TXB8	Το ένατο bit στάλθηκε (σε λειτουργία 9-bit)

Η κατάσταση όπου βρίσκεται το USART έχει πολύ μεγάλη σημασία. Αν υποθέσουμε πως θέλουμε να στείλουμε 1 byte πληροφορίας με έναν μικροελεγκτή που λειτουργεί σε συχνότητα 8 Mhz με ρυθμό μετάδοσης 9600 θα χρειαστούμε 1.04 ms. Οι εντολές που μπορεί να εκτελέσει ο μικροελεγκτής σε αυτόν τον χρόνο ανέρχονται περίπου στις 8000.Ο μικροελεγκτής θα έχανε χρόνο αν περίμενε την σειριακή επικοινωνία να ολοκληρωθεί. Ετσι είναι σημαντικό το πρόγραμμα να γνωρίζει την κατάσταση που βρίσκεται η σειριακή πόρτα. Στην περίπτωση που στείλουμε ένα byte χρειάζεται περίπου 1.04 ms να φτάσει στον δέκτη. Τότε το bit RXC τίθεται σε λογικό 1 σηματοδοτώντας την επιτυχία την λήψης. Κατά τον ίδιο τρόπο το UDRE τίθεται σε λογικό 1 όταν ο UDR είναι ελεύθερος και μπορούμε να φορτώσουμε ένα άλλο byte στον UDR για να γίνει η σειριακή μετάδοση. Παρακάτω βλέπουμε τον καταχωρητή κατάστασης του USART τον USCRA

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

Στην συνέχεια βλέπουμε συνοπτικά στον πίνακα την λειτουργία του εκάστοτε bit.

BIT	ΠΕΡΙΓΡΑΦΗ
RXC	Λογικό 1 όταν γίνει λήψη ενός byte.
TXC	Λογικό 1 όταν γίνει αποστολή ενός byte.
UDRE	Λογικό 1 όταν αδειάσει ο καταχωρητής δεδομένων UDR
FE	Λογικό 1 όταν υπάρχει σφάλμα πλαισίου.(όταν το stop bit είναι 0 αντί για 1)
DOR	Λογικό 1 όταν υπάρχει επικάλυψη στον καταχωρητή UDR

PE	Λογικό 1 όταν υπάρχει σφάλμα στο bit ισοτιμίας
U2X	Λογικό 1 αν επιθυμούμε τον διπλασιασμό της ταχύτητας αποστολής
MPCM	Λογικό 1 αν επιθυμούμε λειτουργία πολλαπλής επικοινωνίας

Στην συνέχεια θα δούμε μια εφαρμογή όπου θα είναι συνδεμένος ένας μικροελεγκτής με ένα pc μέσω της θύρας RS232 με την βοήθεια του ολοκληρωμένου MAX232.Όταν ο χρήστης πατήσει στο πληκτρολόγιο του υπολογιστή το πλήκτρο a, θα εμφανίζεται ένα μήνυμα στην οθόνη το οποίο θα ανακτάται από την μνήμη του μικροελεγκτή. Αντίστοιχα αν πατήσει το b θα εμφανίζει ένα άλλο μήνυμα, καθώς και αν πατηθεί οποιοδήποτε άλλο πλήκτρο θα εμφανίζεται ένα τρίτο μήνυμα.

```
#include <mega16.h>
unsigned char qcntr=0,sndcntr=0;
unsigned char queue[50];

char msg1={"path8hke to plhktro a");
char msg2={"path8hke to plhktro b");
char msg3={den path8hke oute to a, oute to b"};

interrupt [USART_TXC] void usart_transmit_isr (void)
{
        If (qcntr != sndcntr)   UDR=queue[sndcntr++];
}

void sendmsg (char*s)
{
        qcntr=0;
        sndcntr=1;
        queue[qcntr++]=0x0d;
        queue[qcntr++]=0x0a;
        while (*s) queue[qcntr++]=*s++;
        UDR=queue[0];
}
void main(void)
```

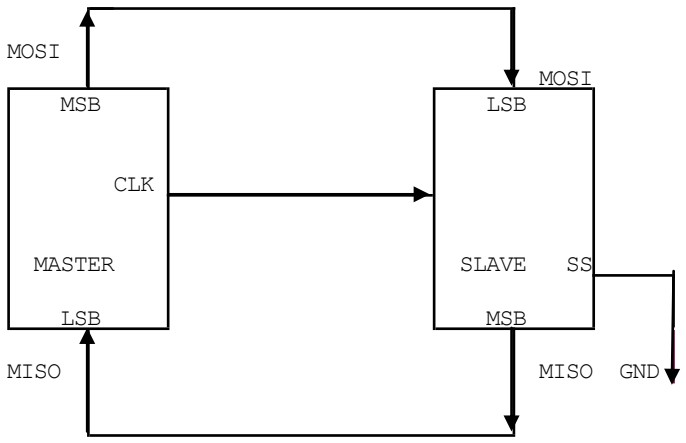
```
{
        char ch;            //μεταβλητή όπου αποθηκεύεται  ο εισερχόμενος χαρακτήρας
       UCSRA=0x00;  /*αρχικοποίηση του USART */
        UCSRB=0x58; //ενεργοποίηση του δέκτη, του πομπού και των διακοπών αποστολής
       UBRRH=0x00;
       UBRRL=0x26;
#asm("sei")             //ενεργοποιούμε τις διακοπές.
while(1)
{
         if  (UCSRA & 0x80)
                 {
                         ch=UDR;
                         switch (ch)
                         {
                                 case 'a':
                                 sendmsg (msg1);
                                 break;
                                 case 'b':
                                 sendmsg (msg2);
                                 break;
                                 default:
                                 sendmsg (msg3);
                         }
                 }
}
}
```

ΤΟ ΠΡΩΤΟΚΟΛΛΟ SPI

Το πρωτόκολλο SPI είναι μια μορφή σειριακής επικοινωνίας που υποστηρίζεται και από τους μικροελεγκτές. Η μετάδοση των δεδομένων γίνεται με σύγχρονο τρόπο. Αυτό σημαίνει πως ο πομπός και ο δέκτης μοιράζονται το ίδιο ρολόι χρονισμού έτσι ώστε να

ανιχνεύονται τα bit στον δέκτη. Συνήθως το πρωτόκολλο SPI χρησιμοποιείται σε πολύ κοντινές αποστάσεις επικοινωνίας, όπως η επικοινωνία μεταξύ μικροελεγκτών που βρίσκονται στην ίδια πλακέτα. Διαφέρει με το πρωτόκολλο USART το οποίο χρησιμοποιείται σε μακρινότερες επικοινωνίες όπως η σύνδεση ενός μικροελεγκτή με ένα PC. Το πρωτόκολλο SPI είναι κατασκευασμένο έτσι ώστε να παρέχει σχετικά μεγάλη ταχύτητα επικοινωνίας, επικοινωνία σε μικρή απόσταση χρησιμοποιώντας παράλληλα ελάχιστα pin στον μικροελεγκτή

Η SPI επικοινωνία επιτυγχάνεται με δύο συσκευές, μια master και μια slave. Και οι δυο συσκευές στέλνουν και λαμβάνουν δεδομένα ταυτόχρονα, όμως η master συσκευή είναι υπεύθυνη στην παραγωγή του σήματος χρονισμού για την μεταφορά των δεδομένων. Έτσι η master συσκευή έχει τον έλεγχο της ταχύτητας επικοινωνίας της μεταφοράς δεδομένων. Παρακάτω βλέπουμε την σύνδεση μεταξύ της master και της slave συσκευής.



Η master συσκευή παρέχει το σήμα χρονισμού (CLK) και τα 8 bit δεδομένων τα οποία ολισθαίνουν μέσω του MOSI pin. Τα bit εισέρχονται στην slave συσκευή, ένα ένα σε κάθε χτύπο του ρολογιού. Ταυτόχρονα από την slave συσκευή μέσω του MISO pin ολισθαίνουν 8 διαφορετικά bit δεδομένων με προορισμό την master συσκευή. Υπάρχει δυνατότητα να συνδεθούν πολλές συσκευές μαζί χρησιμοποιώντας τον ίδιο δίαυλο SPI. Κάθε συσκευή που είναι συνδεμένη πάνω στον δίαυλο μπορεί να γίνει slave γειώνοντας το pin SS. Συνήθως τα σήματα SS συνδέονται μέσω ενός αποκωδικοποιητή στο master ο οποίος ορίζει ποια συσκευή θα είναι slave.

Για τον έλεγχο του SPI οι μικροελεγκτές AVR διαθέτουν τον καταχωρητή SPCR τον οποίο βλέπουμε παρακάτω.

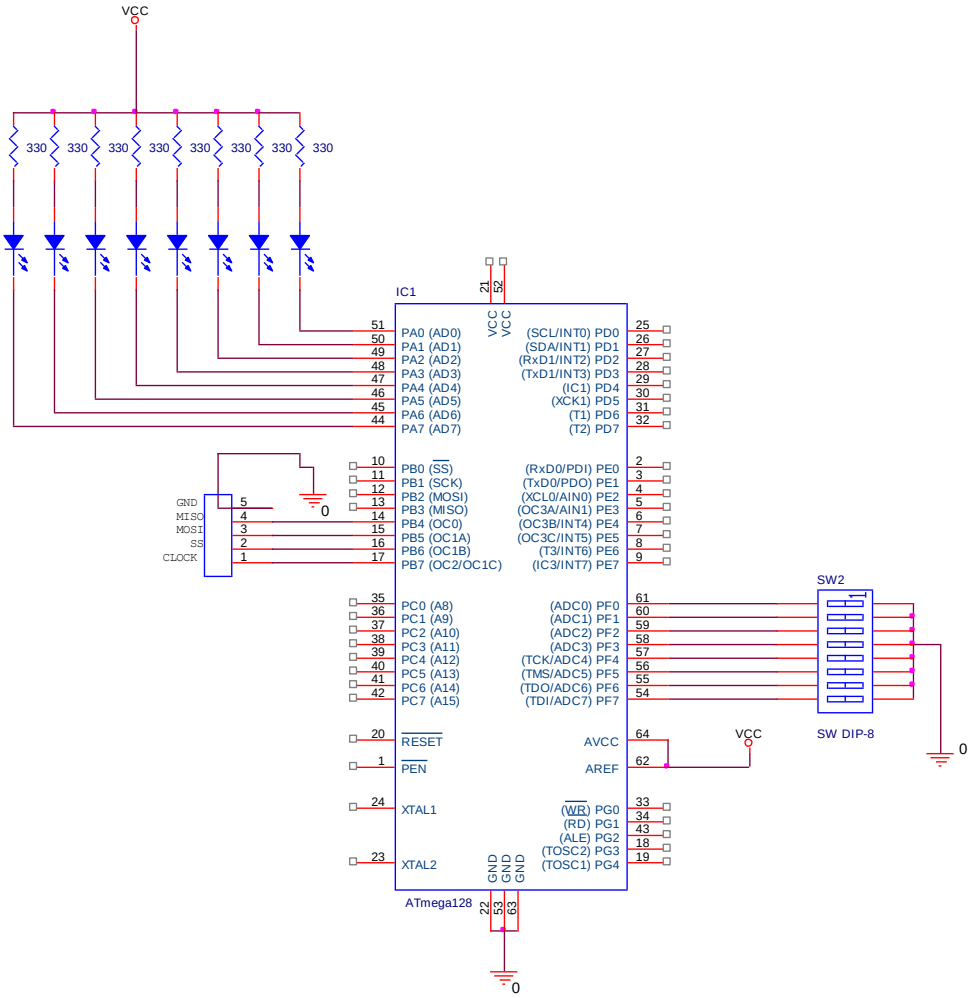
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

BIT	ΠΕΡΙΓΡΑΦΗ
SPIE	Ενεργοποίηση διακοπής από το SPI
SPE	Ενεργοποίηση των λειτουργιών SPI
DORD	DATA ORDER. Όταν είναι 0 αναγκάζει το MSB της ψηφιακής λέξης να μεταδοθεί πρώτο
MSTR	Master/slave select .Όταν βρίσκεται σε λογικό 1, η συσκευή είναι master
CPOL	Πολικότητα του σήματος χρονισμού
CPHA	Φάση του κυκλώματος χρονισμού
SPR1	Ρυθμός ρολογιού του SPI
SPR0	Ρυθμός ρολογιού του SPI

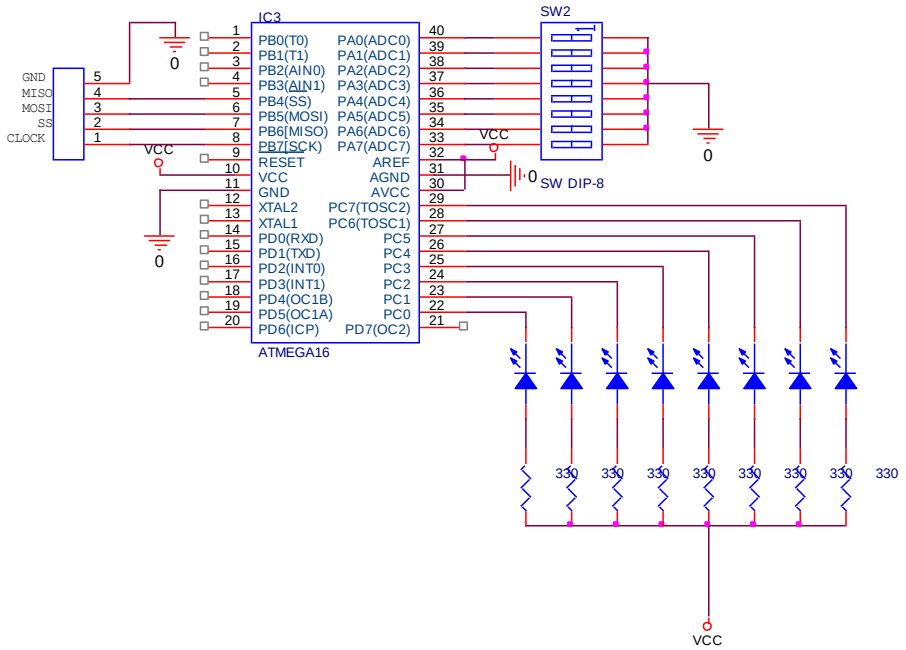
SPR1	SPR0	SCLK
0	0	System clock / 4
0	1	System clock / 16
1	0	System clock / 64
1	1	System clock / 128

Στην συνέχεια θα δούμε μια εφαρμογή όπου θα διασυνδέσουμε δυο μικροελεκτές με το πρωτόκολλο SPI με σκοπό την μετάδοση δεδομένων. Ως master συσκευή θα χρησιμοποιήσουμε τον atmega128 και ως slave συσκευή τον atmega16. Στην porta του atmega16 έχουμε συνδέσει μια σειρά από οκτώ switches. Σκοπός μας είναι ανάλογα με την δυαδική ακολουθία που θα σχηματίζουμε εναλλάσσοντας την κατάσταση των διακοπτών , να γίνεται η επικοινωνία μέσω SPI με τον atmega128 και να ανάβουνε τα αντίστοιχα led που βρίσκονται στην porta. Αντίστοιχα οι διακόπτες που είναι συνδεμένοι στην Portf του atmega128 θα ανάβουν τα led που βρίσκονται στην portc του at mega16. Η σύνδεση μεταξύ των δύο μικροελεγκτών μπορεί να γίνει είτε απευθείας είτε με connector. Στην συνέχεια βλέπουμε τα σχηματικά διαγράμματα του atmega16 και του atmega128.

Το κύκλωμα για τον ATMEGA128



Το κύκλωμα για τον ATMEGA16



Για την λειτουργία των διακοπτών έχουμε ενεργοποιήσει τις εσωτερικές pull up αντιστάσεις στους μικροελεγκτές. Στην συνέχεια βλέπουμε το software για τον atmega128 και τον atmega16

Ο κώδικας για τον ATMEGA128

```
#include <mega128.h>
#define SPI_output_data PORTA
#define SPI_input _data PINF
Interrupt [SPI_STC] void spi_isr(void)        //Ρουτίνα διακοπής του SPI
{
        PORTB.0=1;              //θέτουμε το SS σε υψηλό δυναμικό
        SPI_output_data=~SPDR;  //διαβάζουμε τα δεδομένα που εστάλησαν
        PORTB.0=0;              //θέτουμε το SS σε χαμηλό δυναμικό
        SPDR=SPI_input_data;    //διαβάζουμε τα δεδομένα που λήφθησαν
}
Void main(void)
{
        PORTA=0XFF;             //ξεκινάμε με τα led σβηστά
        DDRA=0XFF;              //θέτουμε την πόρτα ως έξοδο
        PORTB=0X01;             //ξεκινάμε με το SS σε υψηλό δυναμικό
        DDRB=0X07;              //ss,sck και το MOSI ως έξοδος
        PORTF=0XFF;             //Ενεργοποίηση των pull up αντιστάσεων
        SPCR=0XD1;              //ρύθμιση για clock=375Khz,ενεργοποίηση του SPI καθώς
        SPSR=0X00;              //και των διακοπών λόγο του SPI

        #asm
        In r30,spsr     // παρέχεται από τον μεταγλωττιστή για καθαρισμό της σημαίας
        In r30,spdr     // των διακοπών του SPI
        #endasm
        #asm("sei")             //ενεργοποίηση των διακοπών
        SPDR=0X00;
        While(1);
        }
```

Ο κώδικας για τον ATMEGA16

```
#include <mega16.h>
#define SPI_output_ata PORTC
#define SPI_input_data PINA
Interrupt [SPI_STC]    void spi_isr(void)
        {
                SPI_output_data=~SPDR;     //διαβάζουμε τα νέα δεδομένα που ελήφθησαν
                SPDR=SPI_input_data;       //φορτώνουμε νέα δεδομένα για αποστολή
        }

Void main(void)
        {
                DDRB=0X40;      //ρύθμιση του sclk,SS ως είσοδο και του MISO ως έξοδο
                DDRC=0XFF;          //όλα τα pins της πόρτας ως έξοδοι
                PORTA=0XFF;         //ενεργοποίηση των pull up αντιστάσεων
                SPCR=0XC1;   //ενεργοποίηση του SPI και των διακοπών του
                             //ορισμός ως slave συσκευή
                #asm
                In r30,spsr      //παρέχεται από τον μεταγλωττιστή για καθαρισμό της σημαίας
                In r30,spdr      //των διακοπών του SPI
                #endasm

                #asm("sei")
                while(1);
        }
```

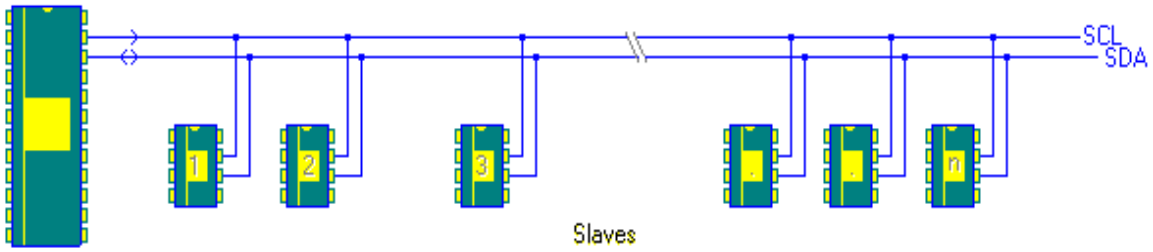
ΤΟ ΠΡΩΤΟΚΟΛΛΟ I^2C

Το I^2C είναι η συντομογραφία του Inter-IC δίαυλου. Δημιουργήθηκε για να διασύνδεει μικροϋπολογιστές και αλλά ICs στο ίδιο PCB.Ο τρόπος διασύνδεσης του στηρίζεται στη σειριακή λειτουργία. Σε ένα μικρουπολογιστικό σύστημα αυτός ο δίαυλος εκτελεί διεργασίες που κανονικά εκτελούνταν από τον παράλληλο δίαυλο.

Η εταιρία Philips της οποίας δημιούργημα είναι ο δίαυλος I^2C παράγει ένα μεγάλο αριθμό από ηλεκτρονικές συσκευές ,οι οποίες τις περισσότερες φορές και για διαφορετικούς λόγους είναι εξοπλισμένες με έναν μικροεπεξεργαστή ή μικροελεγκτή. Επεξεργαστές και περιφερειακά chips χρειάζονται πολλά καλώδια όταν χρησιμοποιείται η παράλληλη επικοινωνία και σαν αποτέλεσμα πολλά pins σύνδεσης που με την σειρά τους απαιτούν αρκετό χώρο πάνω στο PCB.Ο δίαυλος I^2C στην βασική του φόρμα έχει μέγιστη συχνότητα μεταφοράς *10kbyte/sec* η οποία αν και μικρή, είναι αρκετή για τις περισσότερες εφαρμογές. Λόγω του ότι ο δίαυλος I^2C χρησιμοποιεί δύο καλώδια εξοικονομούμε αρκετό χώρο στο PCB. Τα ICs γίνονται ολοένα και μικρότερα σε μέγεθος δίνοντας έτσι την δυνατότητα να μειωθεί ο χρόνος παράγωγης τους και οι κυκλωματικές του διαστάσεις. Ο δίαυλος I^2C έγινε πρότυπο της χαμηλής σε ταχύτητα επικοινωνία πάνω στο ίδιο PCB.

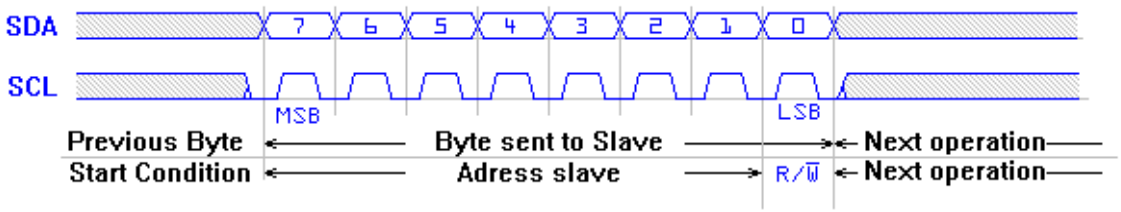
Η ΔΟΜΗ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ I^2C

Το πρωτόκολλο I2C αποτελείται από 2 ενεργά καλώδια και μία σύνδεση γείωσης (ground). Τα ενεργά καλώδια ονομάζονται SDA και SCL και είναι διπλής κατεύθυνσης (bidirectional).Το SDA είναι το Serial DAta line και το SCL το Serial CLock line.κάθε συσκευή που «κρέμεται» από τον δίαυλο έχει μια μοναδική διεύθυνση, χωρίς να έχει σημασία αν πρόκειται για μικροελεγκτή, μνήμη, αισθητήριο η οθόνη υγρών κρυστάλλων. Κάθε συσκευή μπορεί να ενεργεί είτε ως master, είτε ως slave και αντίστοιχα να λαμβάνει η να στείλει εντολές ανάλογα με τις απαιτήσεις του συστήματος που σχεδιάζουμε .Κάθε φορά μόνο μία συσκευή που βρίσκεται στον δίαυλο μπορεί να λειτουργεί ως master.Η συσκευή που λειτουργεί ως master θα εκδώσει αρχικά μια εντολή START. Με την παρουσία της εντολής αυτής στον δίαυλο, όλες οι συσκευές slave θα περιμένουν να στείλει η master συσκευή δεδομένα. Μετά το πέρας της εντολής ένα σήμα STOP θα εκδοθεί από



την master συσκευή και ο δίαυλος θα είναι διαθέσιμος και πάλι (κατάσταση idle) σε όποια συσκευή χρειαστεί. Μετά την εντολή start η master συσκευή στέλνει στην λεωφόρο την

διεύθυνση της συσκευής που θέλει να έχει πρόσβαση καθώς και ένα αναγνωριστικό byte για το αν ζητά εγγραφή ή ανάγνωση από την συγκεκριμένη συσκευή. Οι συσκευές slave συγκρίνουν την διεύθυνση αυτή με την δική τους διεύθυνση και αν δεν είναι η δικιά τους, απλά περιμένουν το STOP σήμα από την master



Στο παραπάνω σχήμα βλέπουμε τα bit διευθύνσεων που ξεκινούν από το MSB και σταματούν στο bit 1.Άρα έχουμε 7 bit για διευθυνσιοδότηση, δηλαδή 2^7=128 συσκευές. Το bit 0 αντιπροσωπεύει την εντολή για εγγραφή ή ανάγνωση Read=1,Write=0

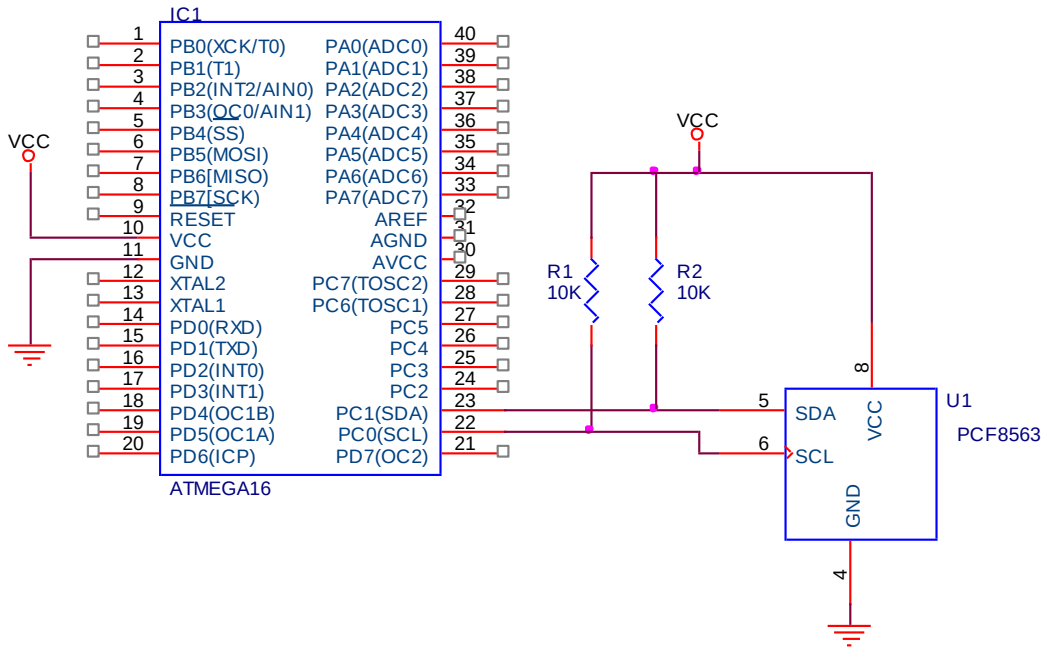
Η μεγάλη ανάπτυξη του I2C και ο περιορισμός των συσκευών που προκύπτει από την 7 bit διευθυνσιοδότηση, λύνεται με την απόδοση διευθύνσεων 10bit.Αυτό επιτυγχάνεται στέλνοντας 2 byte όπου το πρώτο έχει την μορφή 11110XX όπου XX τα 2 πιο σημαντικά bit της διεύθυνσης. Στην συνέχεια στέλνεται και το επόμενο byte με τα υπόλοιπα 8 bit της διεύθυνσης.

Όταν η διεύθυνση αναγνωριστεί από την slave ως δική της, τότε η τελευταία στέλνει ένα σήμα ACK(ACKNOWLEDGED) για να αντιληφθεί η master πως η slave είναι έτοιμη να δεχθεί ή να στείλει δεδομένα. Αυτό γίνεται θέτοντας η slave το SDA σε high στον ένατο παλμό του SCL. Μετά το πέρας τον δεδομένων η master συσκευή θα στείλει ένα STOP σήμα έτσι ώστε να έρθει ο δίαυλος σε κατάσταση αδράνειας(idle).

Τα σήματα χρονισμού του διαύλου μας παρέχονται από τον μεταγλωττιστή και δεν απασχολούν άμεσα τον προγραμματιστή. Στην συνέχεια θα εξετάσουμε πως μπορούμε να διαβάσουμε την ώρα που μας παρέχεται από ένα ρολόι πραγματικού χρόνου το PCF8563TD το οποίο λειτουργεί με το πρωτόκολλο I^2C

```
#include <mega16.h>
#asm
        .equ    __i2c_port=0x15 ;portc
        .equ    __sda_bit=1
        .equ    __sc1_bit=0
#endasm
#include<pcf8563.h>
```

```
Void main(void)
{
        Unsigned char hout,minute,second;
DDRB=0Xff;
i2c_init();
rtc_init(1,RTC_CLKOUT_OFF,RTC_TIMER_OFF);
while(1)
        {
        Rtc_get_time(&hour,&minute,&second);
PORTB=~second
        };
}
```

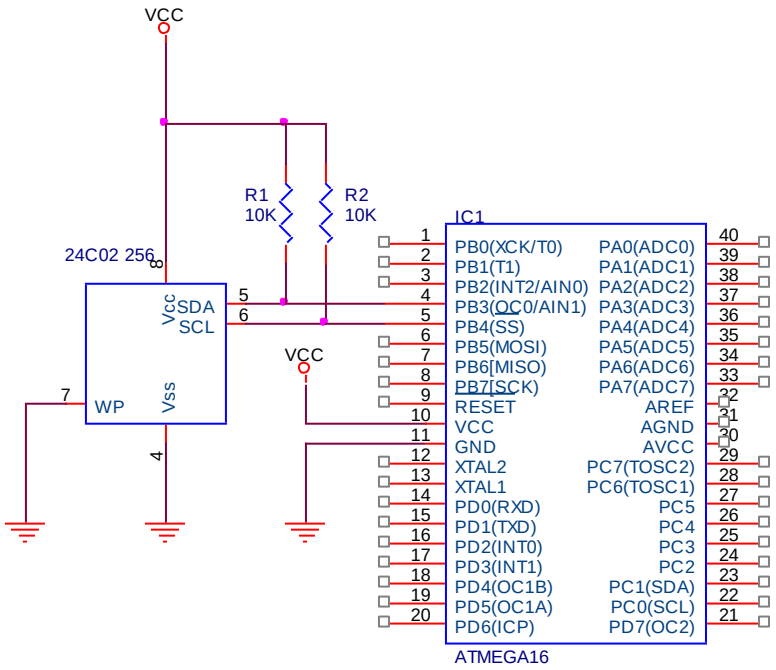


Στην συνέχεια θα δούμε πως μπορούμε να έχουμε πρόσβαση σε μια μνήμη epprom της ATMEL (την 24C02 256) που στηρίζεται στο πρωτόκολλο I^2C με τον μικροελεγκτή ATMEGA16. Τα σήματα SDA και SCL βρίσκονται στην PORTB.

```
#asm
.equ __i2c_port=0x18        //Η λεωφόρος I2C είναι συνδεμένη στην PORTB στα bits
.equ __sda_bit=3            //3 και 4
.equ __scl_bit=4
```

```
#endasm
#include <mega16.h>
#include <i2c.h>                //συμπεριλαμβάνουμε το πρωτόκολλο I2C
#include <delay.h>              // συμπεριλαμβάνουμε τις ρουτίνες καθυστέρησης.
#define EEPROM_BUS_ADDRESS 0xa0      //
/* διαβάζουμε ένα byte από την EEPROM */
unsigned char eeprom_read(unsigned char address) {
unsigned char data;
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS);
i2c_write(address);
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS | 1);
data=i2c_read(0);
i2c_stop();
return data;
}
/* εγγραφή ενός byte στην EEPROM */
void eeprom_write(unsigned char address, unsigned char data) {
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS);
i2c_write(address);
i2c_write(data);
i2c_stop();


/* αναμονή 10ms για την ολοκλήρωση της εγγραφής στην EPPROM */
delay_ms(10);
}
void main(void) {
unsigned char i;
DDRD = 0xff;
PORTD = 0x00;
delay_ms(1000);
```

```
/* initialize the I2C bus */
i2c_init();
/* write the byte 55h at address AAh */
eeprom_write(0xaa,0x55);
/* read the byte from address AAh */
i=eeprom_read(0xaa);
PORTD = i;
while (1); /* loop forever */
}
```



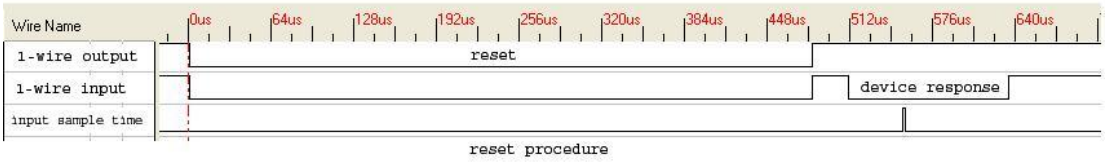
ΤΟ ΠΡΩΤΟΚΟΛΛΟ 1 WIRE

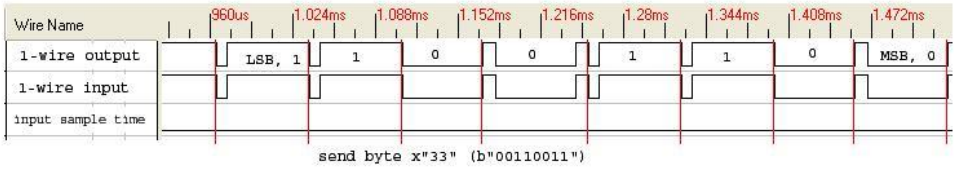
Το πρωτόκολλο I wire είναι ένας δίαυλος επικοινωνίας που αναπτύχθηκε από την Dallas Semiconductor το οποίο παρέχει χαμηλής ταχύτητας επικοινωνία με δίαυλο ένα απλό καλώδιο. Το πρωτόκολλο 1 wire είναι παρόμοιο με το I^2C , αλλά με χαμηλότερους ρυθμούς μετάδοσης από αυτό. Χρησιμοποιείται για να διασύνδεει συσκευές χαμηλής ισχύος και χαμηλής ταχύτητας όπως ψηφιακά θερμόμετρα.

Το πλεονέκτημά του είναι πως χρησιμοποιεί μόνο 1 καλώδια για την επικοινωνία του και γείωση. Για να επιτευχτεί αυτό το ολοκληρωμένο κύκλωμα εμπεριέχει έναν πυκνωτή 800pF, έτσι ώστε να είναι δυνατή η τροφοδοσία του μέσω του διαύλου δεδομένων. Παράλληλα μια αντίσταση pull up των 4.7K εφαρμόζεται στον δίαυλο έτσι

ώστε να δίνει 5 volt στο κανάλι επικοινωνίας και ταυτόχρονα να στέλνει την απαιτούμενη ισχύ σε όλες τις συσκευές που είναι συνδεμένες πάνω στο σύστημα μας.

Η αποστολή δεδομένων ξεκινάει από την master συσκευή στέλνοντας μια αλληλουχία bit. Αρχικά στέλνεται ένας παλμός αρχικοποίησης ο οποίος κρατάει την γραμμή σε λογικό επίπεδο 0 για 480us. Αυτή η εντολή αρχικοποιεί όλες τις slave συσκευές. Έπειτα κάθε slave συσκευή δηλώνει την ύπαρξή της κρατώντας την γραμμή χαμηλά για τουλάχιστον 60us. Για να στείλουμε λογικό 1 στον δίαυλο 1 wire πρέπει να κρατήσουμε την γραμμή χαμηλά για 10us. Αντίστοιχα για να στείλουμε λογικό 0 πρέπει να κρατήσουμε την γραμμή χαμηλά για 60us.





Παραπάνω βλέπουμε τον χρονισμό των σημάτων ελέγχου αρχικοποίησης και μετάδοσης σε δίαυλο 1 wire.

ΑΙΣΘΗΤΗΡΑΣ ΘΕΡΜΟΚΡΑΣΙΑΣ 1 WIRE (DS18B20)

Το ψηφιακό θερμόμετρο ds18B20 είναι μια συσκευή της Dallas Semiconductor. Βασίζεται στο πρωτόκολλο 1 wire και έχει πολλά ενδιαφέρονται χαρακτηριστικά.

1) Έχει έναν μοναδικό κωδικό μήκους 64 bit αποθηκευμένο στην ενσωματωμένη ROM του κάνοντας έτσι δυνατή την σύνδεση πολλών ίδιων συσκευών πάνω στον ίδιο δίαυλο μιας και η κάθε συσκευή έχει την δική της ταυτότητα.2)
2) Μπορεί επίσης να τροφοδοτηθεί με τάσεις από 3.3 έως 5 Volt, ενώ παράλληλα υποστηρίζει παρασιτική τροφοδοσία όπου η τάση λειτουργίας του αισθητήρα λαμβάνεται μέσω της γραμμής data.3)
3) Η ανάλυση της θερμοκρασίας είναι ελεγχόμενη από τον χρήστη (9-12 bits).

4) Έχει εύρος μέτρησης από -55C^o έως 125C^o

5) Διαθέτει μη πτητική μνήμη με δυνατότητα ορισμού θερμοκρασίας συναγερμού.

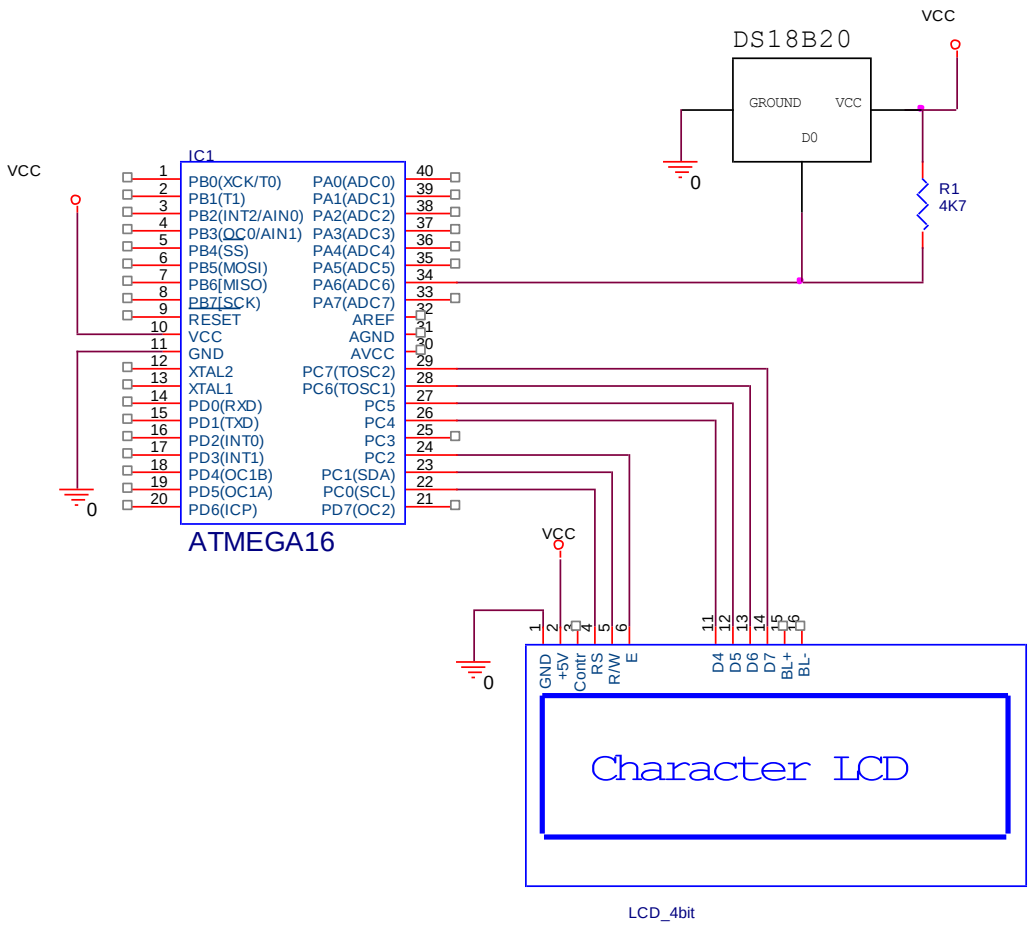
Ο μεταγλωττιστής codevisionAVR εμπεριέχει ρουτίνες οδήγησης για τον συγκεκριμένο αισθητήρα. Έτσι ο έλεγχός του γίνεται γρήγορα χωρίς να χρειάζεται να ανησυχούμε για τα σήματα χρονισμού του. Στην συνέχεια θα κατασκευάσουμε ένα θερμόμετρο που στηρίζεται στον αισθητήρα DS18B20 που θα απεικονίζει την θερμοκρασία του σε μια οθόνη lcd. Η ανάπτυξη του κυκλώματος θα γίνει με τον μικροελεγκτή ATMEGA16.

```
#include <mega16.h>
#asm
.equ __w1_port=0x1b          //ορισμός της πόρτας του μικροελεγκτή που είναι συνδεμένος
.equ __w1_bit=6              //ο αισθητήρας PORTA.6
#endasm
#asm
.equ __lcd_port=0x15         //ορισμός της portc για διασύνδεση με την οθόνη
#endasm
#include <lcd.h>             // ρουτίνες για την οθόνη 2x16
#include <ds18b20.h>         //ρουτίνες για τον αισθητήρα θερμοκρασίας.
#include <delay.h>
#include <stdio.h>
char lcd_buffer[33];         //πίνακας εύρους 33 θέσεων
/*Μέγιστος αριθμός συσκευών που μπορούμε να συνδέσουμε */
#define MAX_DEVICES 8
/* μεταβλητή αποθήκευσης των κωδικών ROM κάθε συσκευής*/
unsigned char rom_code[MAX_DEVICES][9];

main()
{
unsigned char i,j,devices;          //ορισμός μεταβλητών
lcd_init(16);                       //αρχικοποίηση οθόνης
lcd_putsf("CodeVision AVR \n digital thermometer ");
delay_ms(2000);
lcd_clear();
```

```
devices=w1_search(0xF0,rom_code);          //έλεγχος πόσων αισθητήρων είναι συνδεμένοι
sprintf(lcd_buffer,"%u DS18B20\n SYSKEYES BRETHIKAN",devices);
lcd_puts(lcd_buffer);
delay_ms(2000);

/* εκτύπωση του αριθμού της ROM κάθε αισθητήρα */
if (devices)
{
for (i=0;i<devices;i++)
{
sprintf(lcd_buffer,"Device #%u ROM\nCode is:",i+1);
lcd_clear();               //καθαρισμός οθόνης
lcd_puts(lcd_buffer);  //εκτύπωση string
delay_ms(2000);
lcd_clear();
for (j=0;j<8;j++)
{
sprintf(lcd_buffer,"%02X ",rom_code[i][j]);
lcd_puts(lcd_buffer);
if (j==3) lcd_gotoxy(0,1);
};
delay_ms(5000);
};
}
else
while (1);                 //σταματάμε εδώ αν δεν βρεθούν συσκευές.


//Καθορισμός 12 bit ανάλυση για κάθε συσκευή
for (i=0;i<devices;)
if (!ds18b20_init(&rom_code[i++][0],20,30,DS18B20_12BIT_RES))
{                    //ορισμός κατώτερης θερμοκρασίας στους 20 C  και η μέγιστη στους 30
sprintf(lcd_buffer,"lathos arxikopoihshs ston \n aisththra #%u",i);
lcd_clear();
```

```
lcd_puts(lcd_buffer);
while (1); /* σταματάμε εδώ σε λάθος αρχικοποίησης */
};

/* λήψη και εκτύπωση στην οθόνη των θερμοκρασιών*/
while (1)
{
j=1;
for (i=0;i<devices;i++)
{
sprintf(lcd_buffer,"t%u=%.3f\xdfC",j++,ds18b20_temperature(&rom_code[i][0]));
lcd_clear();
lcd_puts(lcd_buffer);
delay_ms(500);
};
};
}
```

Όπως μπορούμε να δούμε στον κώδικά μας, εφόσον βρούμε τον αριθμό των αισθητήρων που είναι συνδεμένοι πάνω στο 1 wire καλώδιο, στην συνέχεια εκτυπώνουμε τον μοναδικό αριθμό κάθε αισθητήρα σε δεκαεξαδική μορφή. Στην συνέχεια αρχικοποιούμε τους αισθητήρες μας, δηλώνουμε την ελάχιστη θερμοκρασία που μπορούμε να έχουμε συναγερμό όταν πέσει κάτω από αυτή,την μέγιστη θερμοκρασία συναγερμού, καθώς και την ανάλυση που θέλουμε να έχουμε. Ταυτόχρονα αρχικοποιούμε τον αισθητήρα και σε περίπτωση λάθους αρχικοποίησης σταματάμε το πρόγραμμα. Το τελευταίο βήμα είναι να διαβάσουμε την θερμοκρασία και να την εκτυπώσουμε με την συνάρτηση lcd_puts. Όπως διαπιστώνουμε ο αριθμός εκτύπωσης είναι κινητής υποδιαστολής (floating) έτσι ώστε να παίρνουμε και το δεκαδικό μέρος της θερμοκρασίας. Στον κώδικά μας μεριμνήσαμε ώστε να είναι δυνατή η σύνδεση μέχρι 8 αισθητήρων. θεωρητικά μπορούμε να συνδέσουμε άπειρους αισθητήρες, μιας και ο κάθε αισθητήρας έχει μοναδικό κωδικό στην ROM του. Στο σχηματικό βλέπουμε συνδεμένο έναν αισθητήρα.

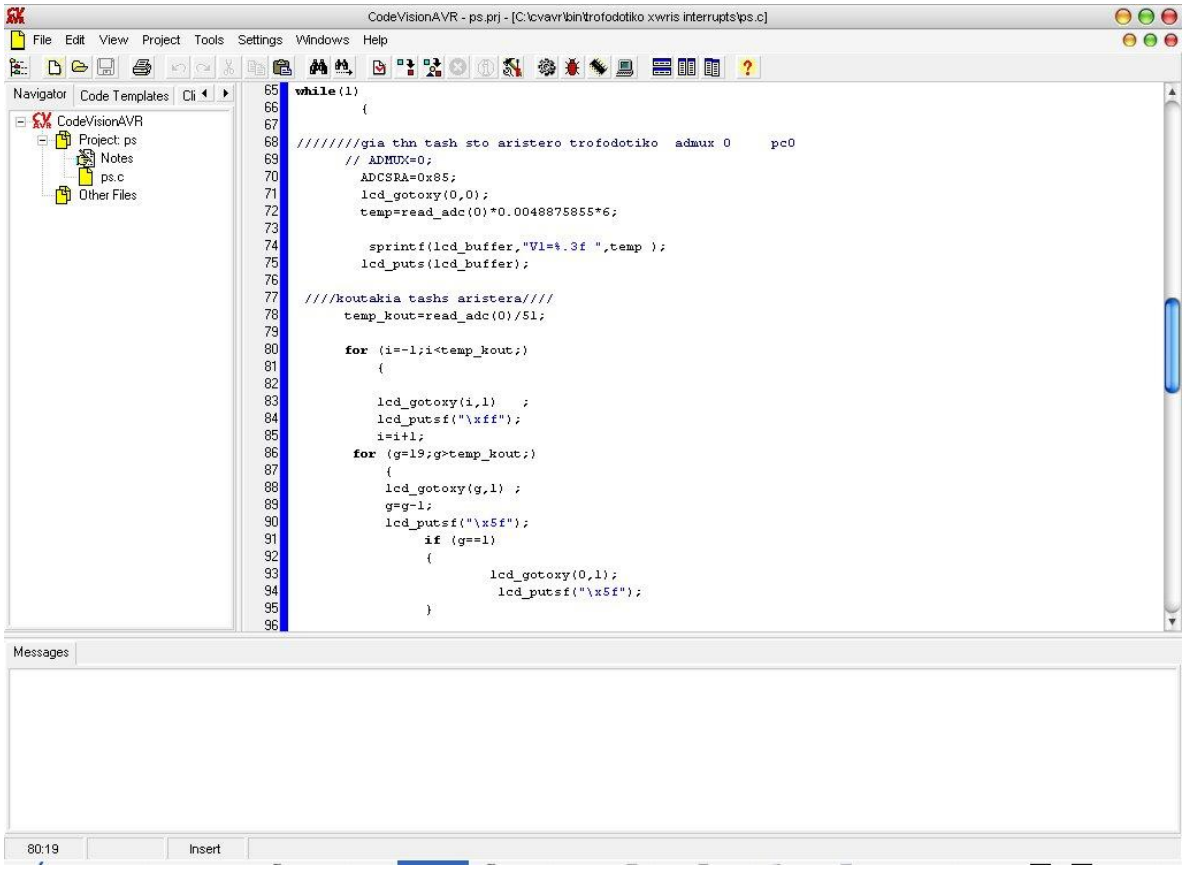


Ο ΜΕΤΑΓΛΩΤΤΙΣΣΤΗΣ CODEVISION

Ο codevision είναι ένας μεταγλωττιστής C για τους AVR από την HP INFOTECH. Το πρόγραμμα τρέχει σε περιβάλλον 32bit κάτω από τα λειτουργικά συστήματα windows 95,98 2000 και XP. Το περιβάλλον εργασίας χαρακτηρίζεται από την φιλικότητα προς τον χρήστη, διαθέτοντας πολλά ενδιαφέρονται χαρακτηριστικά.

Εκτός από την συνηθισμένες βιβλιοθήκες για την C, έχουμε υποστήριξη επιπλέον για

- ✓ Μονάδες οθόνης lcd
- ✓ Το πρωτόκολλο I^2C
- ✓ Τον αισθητήρα θεοκρασίας LM35 της national semiconductor
- ✓ Ρολόγια πραγματικού χρόνου
- ✓ Το πρωτόκολλο 1 wire της Dallas Semiconductor
- ✓ Ψηφιακούς αισθητήρες θερμοκρασίας (DS1820)
- ✓ Το πρωτόκολλο SPI
- ✓ Διαχείριση ενέργειας του μικροελεγκτή
- ✓ Καθυστέρηση προγράμματος (delays)



Επίσης διαθέτει ένα εργαλείο για αυτόματη αρχικοποίηση πολλών συσκευών στους μικροελεγκτές όπως

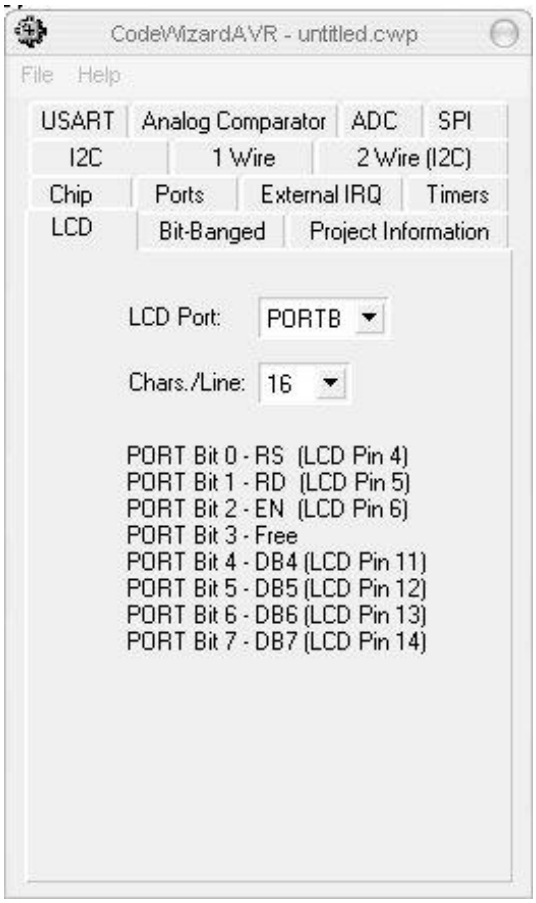
- ✓ Πρόσβαση σε εξωτερική μνήμη
- ✓ Αρχικοποίηση εισόδων εξόδων στον μικροελεγκτή
- ✓ Αρχικοποίηση εξωτερικών διακοπών
- ✓ Αρχικοποίηση των μετρητρών και των χρονιστών
- ✓ Αρχικοποίηση του watchdog timer
- ✓ Αρχικοποίηση του USART
- ✓ Αρχικοποίηση του αναλογικού συγκριτή
- ✓ Αρχικοποίηση του ADC
- ✓ Αρχικοποίηση του SPI
- ✓ Αρχικοποίηση του διαύλου I^2C
- ✓ Αρχικοποίηση οθόνης lcd

Επίσης διαθέτει έναν προγραμματιστή ο οποιός συνεργάζεται με το STK500

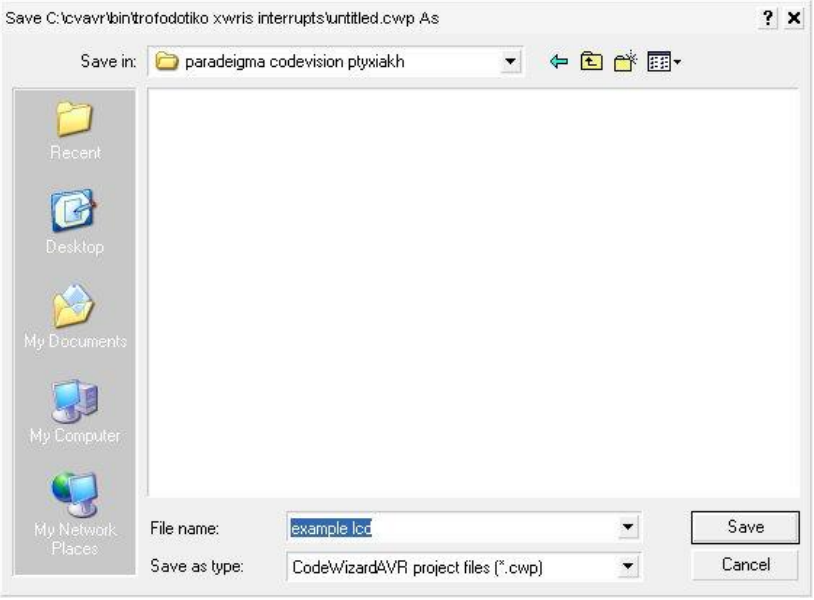
Στην συνέχεια θα δείξουμε πόσο απλό είναι να δημιουργήσουμε ένα πρόγραμμα όπου θα εμφανίζεται ένα μήνυμα σε οθόνη lcd με την βοήθεια του wizard του μεταγλωττιστή.

Αρχικά επιλέγουμε file►new ► project.Το πρόγραμμα μας ρωτάει αν θέλουμε να εκιννήσουμε τον codewizard. Θα απαντήσουμε ναι

Στην συνέχεια θα επιλέξουμε από την καρτέλα ως μικροελεγκτή που θα χρησιμοποιήσουμε τον atmega16 και θα ορίσουμε ως πόρτα για την οθόνη την PORTB,καθώς και τον αριθμό των γραμμών της οθόνης.



Όπως βλέπουμε την στιγμή που επιλέγουμε την πόρτα της οθόνης μας δείχνει τα αντίστοιχα σήματα αυτής σε ποια pin πρέπει να συνδεθούν διευκολύνοντάς μας ακόμα περισσότερο. Στην συνέχεια επιλέγουμε file και generate. Έτσι θα πάρουμε έτοιμες τις απαραίτητες αρχικοποιήσεις ,ελαχιστοποιώντας τον κίνδυνο για λάθος στον κώδικα που θα ευθυνόταν σε κακή αρχικοποίηση. Έπειτα επιλέγουμε file, generate save and exit και εμφανίζεται το παρακάτω παράθυρο όπου μας προτρέπει να σώσουμε το project. Δίνουμε ένα όνομα και πατάμε save.



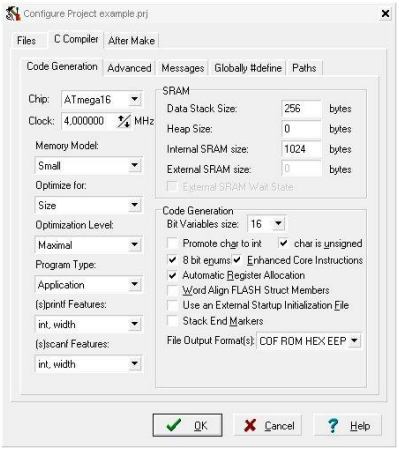
Ο παρακάτω κώδικας δημιουργήθηκε αυτόματα από τον οδηγό του codevision.

```
#include <mega16.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x18 ;PORTB
#endasm
#include <lcd.h>
void main(void)
{
while (1)
      {
      // Place your code here

      };
}
```

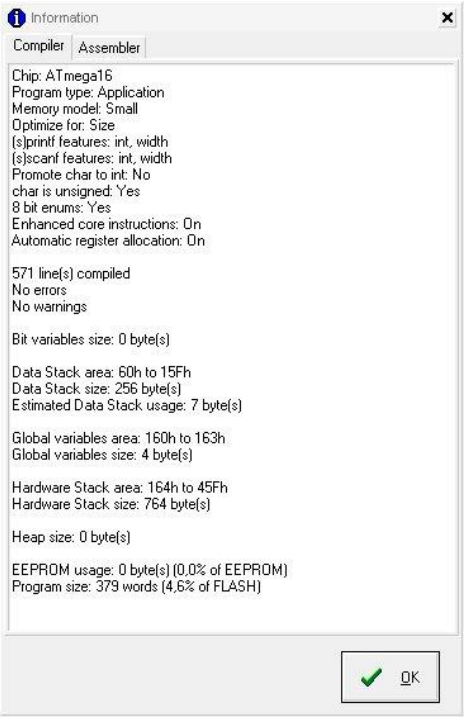
Το μόνο που μένει να κάνουμε είναι να πάμε στο σημείο όπου βλέπουμε να σχόλια place your code here και να γράψουμε την εντολή *sprintf("HELLO");* η οποία θα στείλει το μήνυμα αυτό στην εξωτερική οθόνη. Γίνεται εύκολα αντιληπτό πόσο γρήγορα μπορούμε να αναπτύξουμε μια εφαρμογή με χρήση ενός compiler C. Η ίδια διαδικασία σε assembly θα απαιτούσε επιπλέον προσπάθεια έτσι ώστε να οδηγηθούν τα σήματα της οθόνης σωστά. Επιπλέον η εκσφαλμάτωση του προγράμματος θα ήταν πιο δύσκολη και μια πιθανόν αναβάθμιση θα ήταν επίπονη.

Έπειτα επιλέγουμε από το μενού project → configure και επιλέγουμε την καρτέλα C compiler. Εδώ βλέπουμε τον τύπο του μικροελεγτή, την συχνότητα λειτουργίας του, καθώς

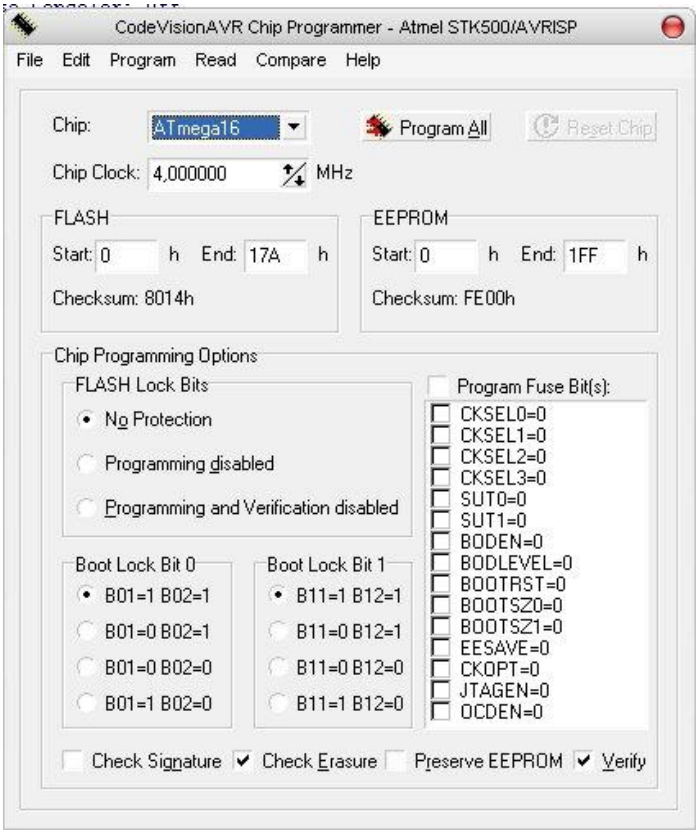
και επιλογές όσον αφορά το πρόγραμμα που θα δημιουργηθεί. Μπορούμε να το βελτιστοποιήσουμε για την ταχύτητα εκτέλεσης του κώδικα ή για το μέγεθος του, ανάλογα με τις ανάγκες μας.



Έπειτα κλείνουμε την καρτέλα και επιλέγουμε από το μενού project→compile και έπειτα project→make. Αν δεν έχουμε κανένα λάθος στον κώδικά μας θα δούμε το εξής παράθυρο



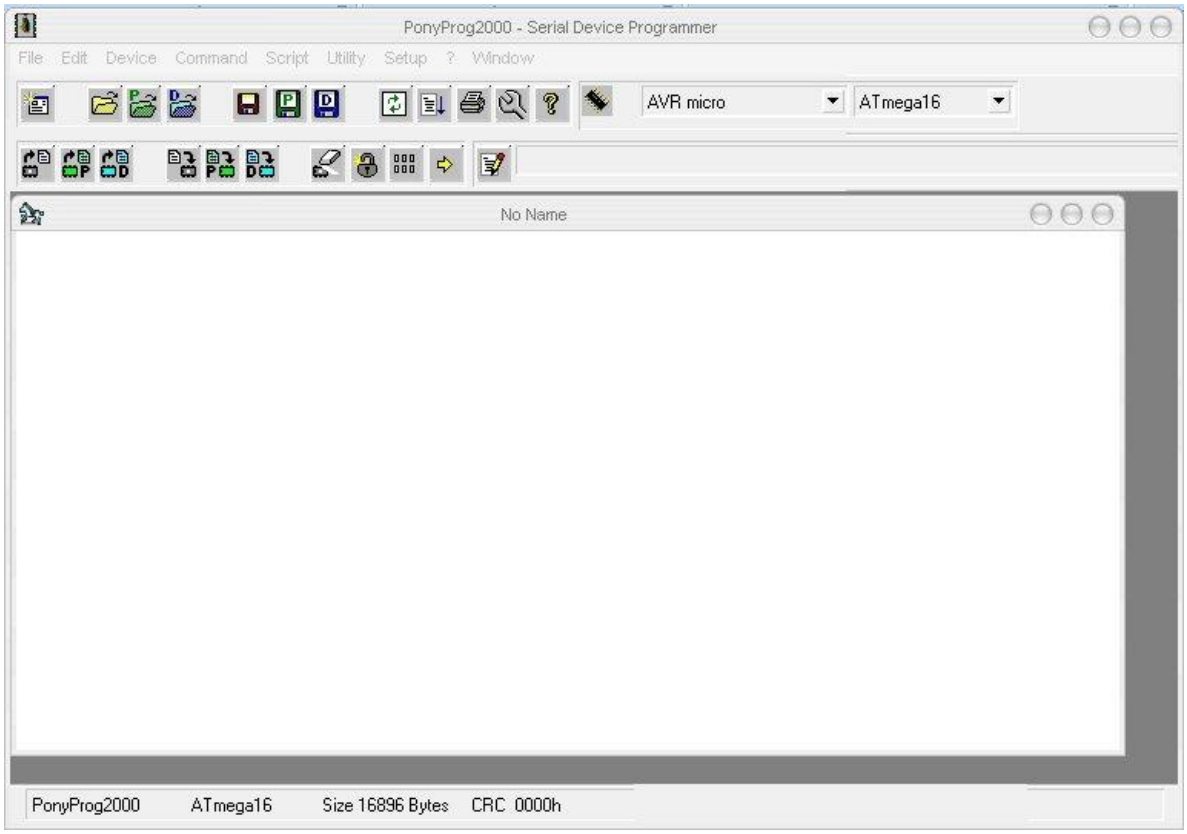
που σημαίνει πως η μεταγλώττιση έγινε σωστά. Ο μεταγλωττιστής διαθέτει ενσωματωμένο έναν προγραμματιστή ο οποίος μπορεί να προγραμματίσει κατευθείαν τον μικροελεγκτή αρκεί να υπάρχει μια συμβατή πλακέτα όπως το STK500, τον AVRISP , τον AVRProg, τον STK200+/300, τον Dontronics DT006, τον Vogel Elektronik VTEC-ISP, τον Futurlec JRAVR ή τον MicroTronics ATCPU/Mega2000



Στην συνέχεια επιλέγουμε program flash εφόσον διαθέτουμε κάποιο από τα παραπάνω αναπτυξιακά συστήματα. Η πλακέτα των εφαρμογών μας δεν συνεργάζεται με τον προγραμματιστή του codevision. Έτσι θα χρησιμοποιήσουμε ένα freeware software το PonyProg2000.

Ο ΠΡΟΓΡΑΜΜΑΤΙΣΤΗΣ PonyProg2000

Ο προγραμματιστής PonyProg2000 είναι ένα software ανοιχτού κώδικα που δημιουργήθηκε από τον Claudio Lanconelli. Είναι εύχρηστο και έχει πολλές λειτουργίες όπως εκτός από το να προγραμματίζει μικροελεγκτές AVR έχει επίσης δυνατότητα να προγραμματίζει μικροελεγκτές της microchip καθώς και μνήμες eeprom. Στην συνέχεια βλέπουμε το περιβάλλον εργασίας του PonyProg2000. Επιλέγουμε από το μενού setup interface setup και αφού είμαστε βέβαιοι πως συνδέσαμε την πλακέτα μας στην σειριακή θύρα του υπολογιστή επιλέγουμε την αντίστοιχη COM που χρησιμοποιούμε. Έπειτα πατάμε το κουμπί probe. Αν η σύνδεση έγινε σωστά, τότε θα λάβουμε το μήνυμα “test ok”. Έπειτα από το μενού device επιλέγουμε avr micro και atmega16. Στη συνέχεια επιλέγουμε από το μενού file → open device file και επιλέγουμε το αρχείο HEX από την θέση που το έχουμε αποθηκευμένο. Έπειτα επιλέγουμε Command και Program.



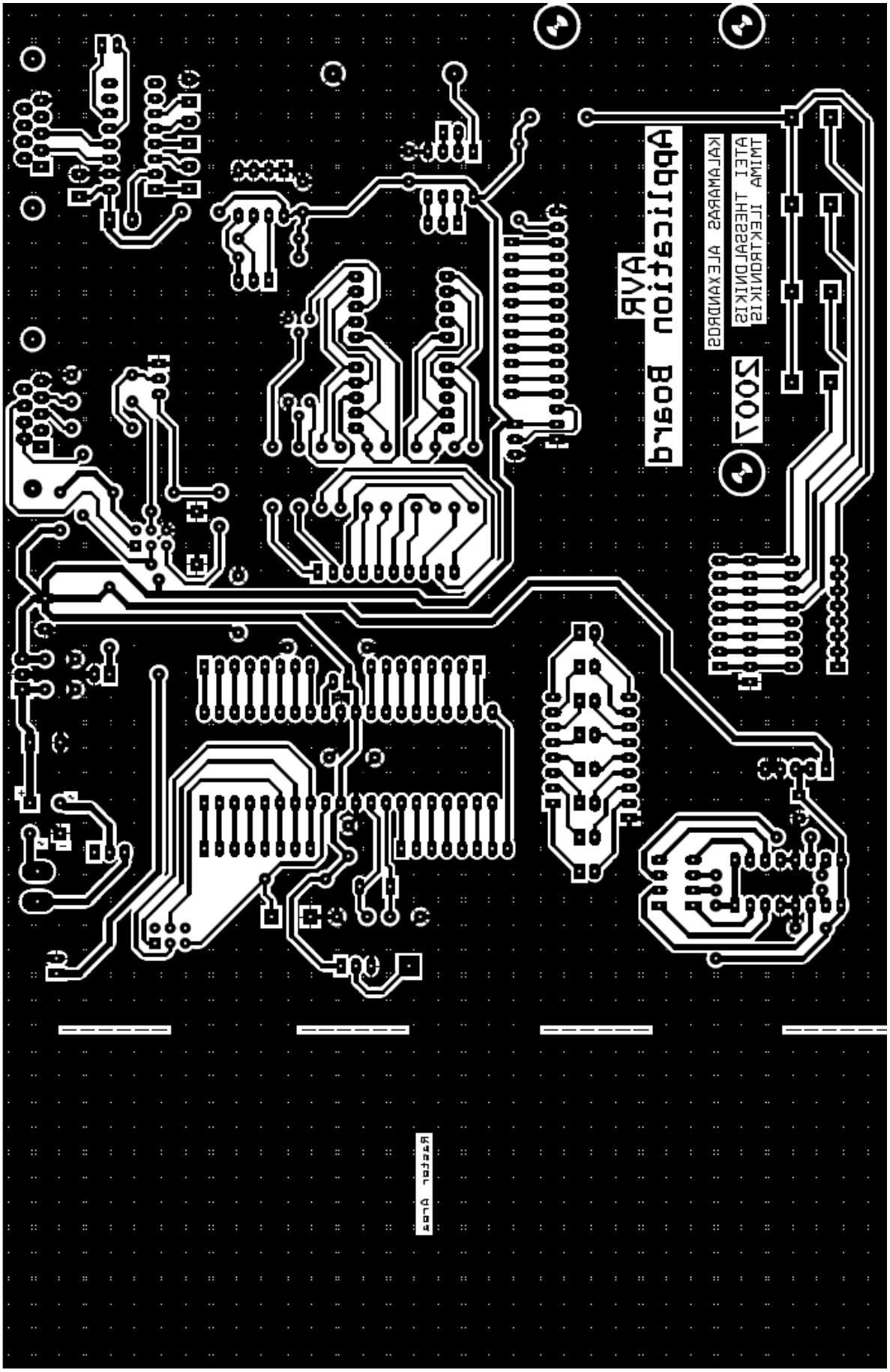
Η ΠΛΑΚΕΤΑ ΕΦΑΡΜΟΓΩΝ ΓΙΑ ΤΟΥΣ AVR

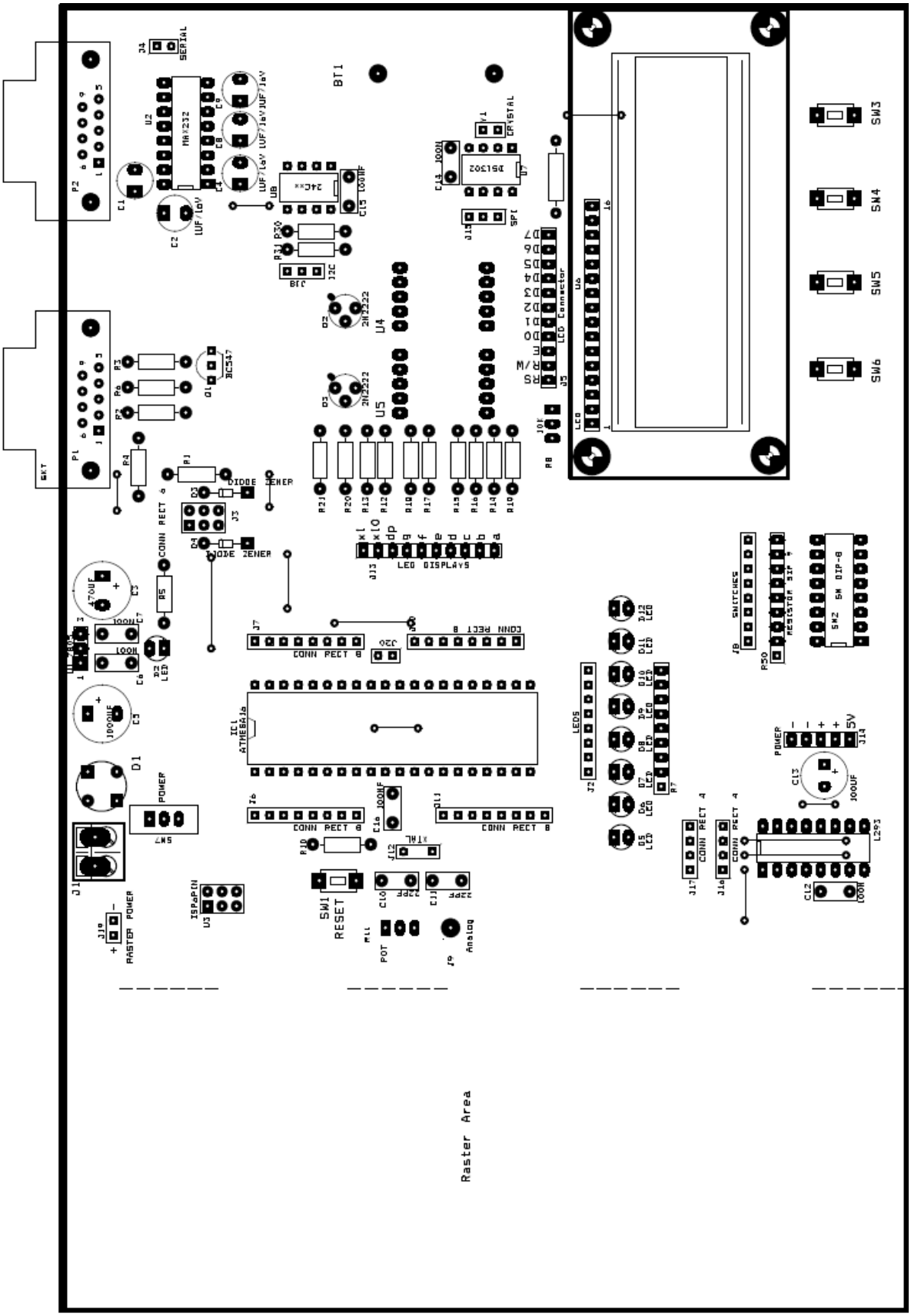
Η πλακέτα που βλέπουμε παρακάτω αναπτύχθηκε έτσι ώστε να είναι δυνατός ο προγραμματισμός του μικροελεγκτή atmega16, καθώς επίσης και όλων των μικροελεγκτών AVR που υποστηρίζουν In System Programming (ISP). Ο προγραμματισμός γίνεται με το PonyProg2000 τον οποίο μπορούμε να κατεβάσουμε δωρεάν από τον δικτυακό τόπο http://www.lancos.com. Η πλακέτα μας έχει τις εξής μονάδες ενσωματωμένες :

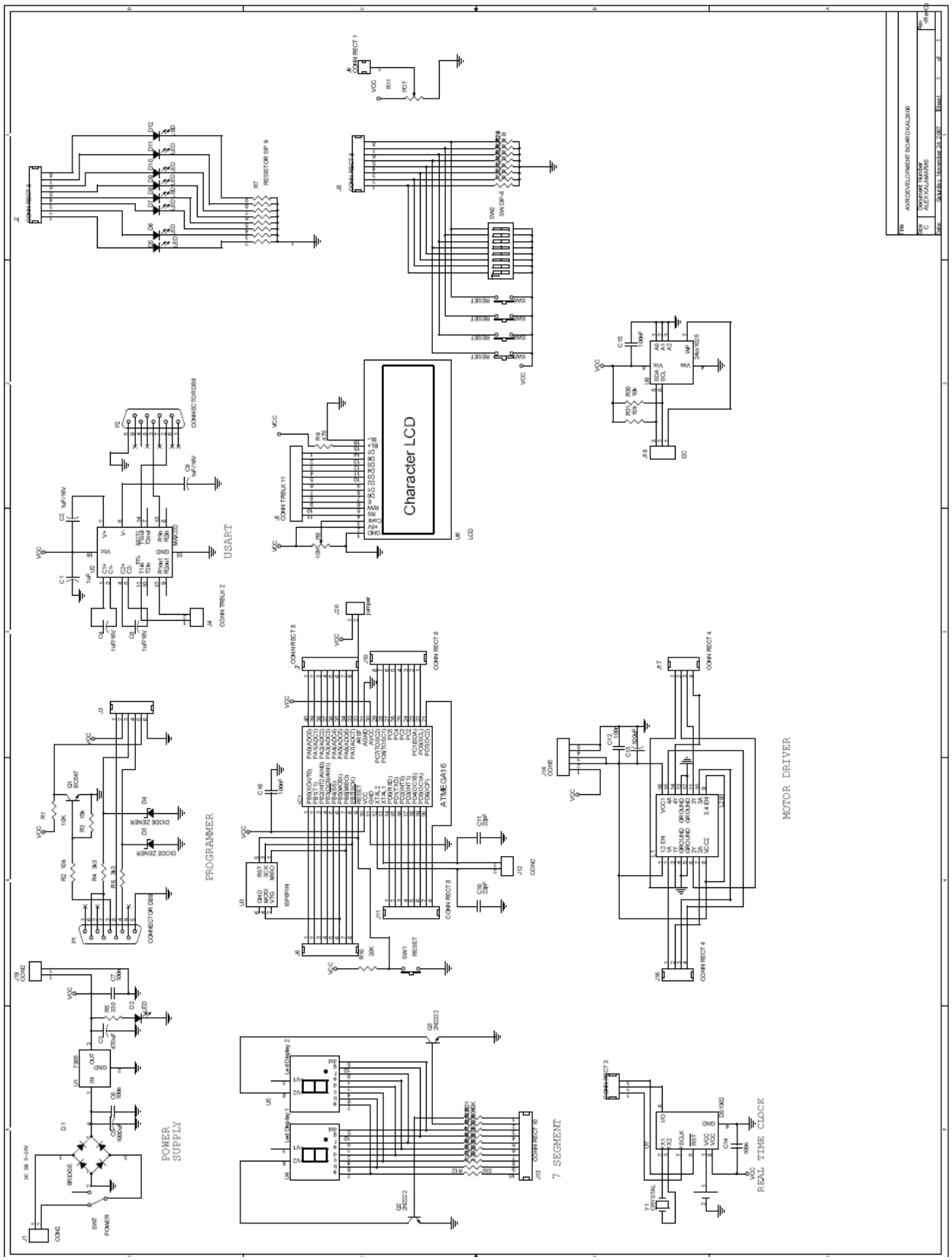
- Τον μικροελεγκτή atmega16 με socket για εξωτερικό κρύσταλλο και επιλογή για την τάση αναφοράς Vref καθώς και ένα button για reset
- Έναν connector των 6 pin για προγραμματισμό συσκευών AVR με ISP.
- Μια θύρα RS232 για επικοινωνία με USART με ηλεκτρονικό υπολογιστή
- Μια θύρα RS232 για τον προγραμματισμό των συσκευών μέσω JTAG
- Μια σειριακή μνήμη που λειτουργεί με το πρωτόκολλο I^2C
- Ένα ρολόι πραγματικού χρόνου της DALLAS SEMICONDUCTOR το DS1302

- Μια μονάδα με δύο led displays κοινής καθόδου τα οποία μπορούν να λειτουργήσουν με πολύπλεξη
- Μια οθόνη lcd 2x16
- Μια μονάδα οδήγησης συσκευών ισχύος με το L293D με την οποία έχουμε δυνατότητα να οδηγήσουμε επαγωγικά φορτία μέχρι 2A από 5 μέχρι 36V, καθώς επίσης και την δυνατότητα να τροφοδοτήσουμε την μονάδα με εξωτερικό τροφοδοτικό.
- 8 διακόπτες σε μορφή dip switch. Επίσης τέσσερα push button τα οποία συνδέονται παράλληλα με τους τέσσερις τελευταίους διακόπτες
- 8 Led τα οποίο λειτουργούν με την θετική λογική
- Ένα ράστερ καθώς και ένα ποτενσιόμετρο όπου μπορούμε να έχουμε τάσεις από 0 μέχρι 5 Volt κατά επιλογή.

Στην συνέχεια βλέπουμε την κάτω και την άνω όψη της πλακέτας των εφαρμογών μας καθώς και το σχηματικό διάγραμμα των διασυνδέσεων των υλικών της. Ο σχεδιασμός της πλακέτας έγινε με το σχεδιαστικό πρόγραμμα ORCAD 9.2


AVR
application board
2007
ΣΤΥΛΙΑΝΟΣ ΚΑΛΑΜΑΡΑΣ ΑΛΕΞΑΝΔΡΟΣ
ΣΤΥΛΙΑΝΟΣ ΤΗΣΣΑΛΟΝΙΚΗΣ
ΑΓΕΙ ΠΑΣΣΑΛΟΝΙΚΗ


SERIAL
MAX232
BT1
CRYSTAL
DS1302
24Cxx
2N2222
U4
U5
LCD Connector
SW3
SW4
SW5
SW6
BC547
DIODE ZENER
CONN RECT 8
LED DISPLAYS
ATMEGA16
LEDS
SWITCHES
RESISTOR SIP 4
SW DIP-8
POWER
L293
SW7
RASTER POWER
ISPaPIN
SW1
RESET
POT
Analog
Raster Area


Character LCD
USART
PROGRAMMER
POWER SUPPLY
7 SEGMENT
MOTOR DRIVER
REAL TIME CLOCK
ATMEGA16
AVRDEVELOPMENT BOARD(AL3)00
Document Number
ALEX KALAMARIS

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Embedded C Programming and the Atmel AVR 2nd edition 2007, Richard H. Barnet, Sarah Cox, Larry O'Cull.
2. Προγραμματίζοντας τον Μικροελεγκτή AVR. Εκδόσεις Τζιόλα Dhananjay V Gardreo, 2001.
3. Μάθετε την C, δεύτερη έκδοση, Tony Zhang. Εκδόσεις Γκιούρδας,2000
4. Μικρουπολογιστές-Μικροελεγκτές.Αρχιτεκτονική-Προγραματισμός-Εφαρμογές, Πογαρίδης Δημήτρης, Εκδόσεις ΙΩΝ 2002
5. Μικρουπολογιστές 1. Σημειώσεις εργαστηρίου του τμήματος ηλεκτρονικής του ΑΤΕΙ Θεσσαλονίκης
6. Μικρουπολογιστές 2. Σημειώσεις εργαστηρίου του τμήματος ηλεκτρονικής του ΑΤΕΙ Θεσσαλονίκης
7. Cad και Ηλεκτρονική Κατασκευή. Σημειώσεις εργαστηρίου του τμήματος ηλεκτρονικής του ΑΤΕΙ θεσσαλονίκης
8. AVR STK500 user manual
9. www.atmel.com
10. www.avrfreaks.net
11. www.serasidis.gr
12. http://users.ntua.gr/el01741/MyWebPage/Contructions/digital/bipstepper.htm
13. http://www.lancos.com/siprogsch.html
14. www.hlektronika.gr
15. http://embedded-inet.com/
16. http://www.mikrocontroller.com/de/einleitung.php
17. http://www.avr-asm-tutorial.net/
18. http://www.avrtutor.com/

ΠΑΡΑΡΤΗΜΑ

ATMEGA16

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 16K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 1K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega16L
  - 4.5 - 5.5V for ATmega16
- Speed Grades
  - 0 - 8 MHz for ATmega16L
  - 0 - 16 MHz for ATmega16

8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

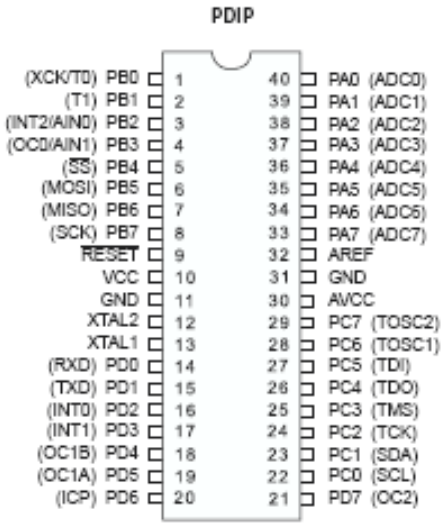
ATmega16
ATmega16L

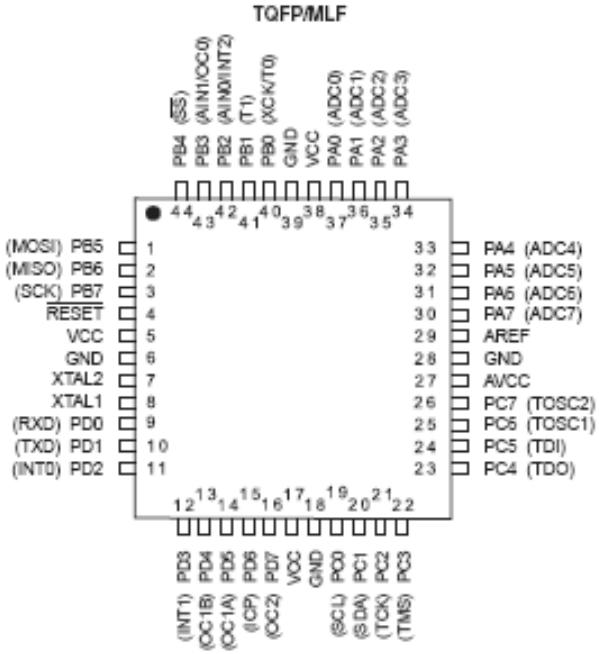
Preliminary

Rev. 2466E–AVR–10/02

Pin Configurations

Figure 1. Pinouts ATmega16





Disclaimer

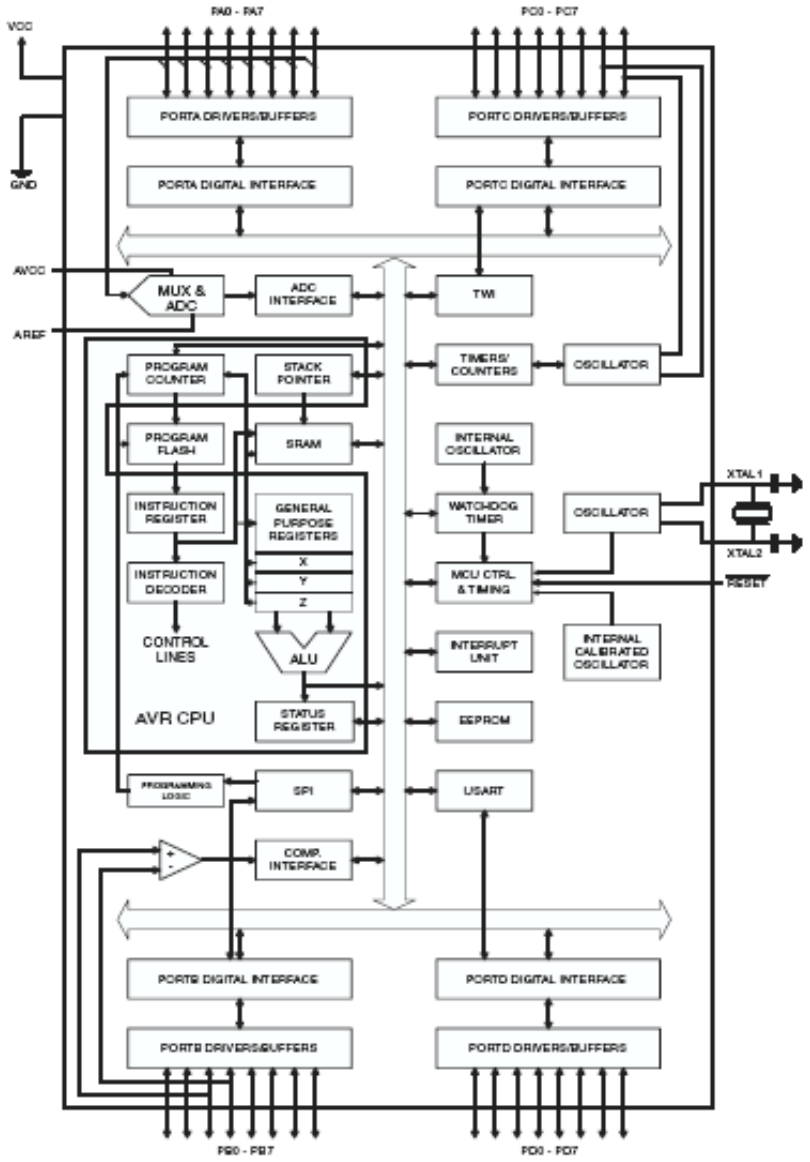
Typical values contained in this data sheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

Overview

The ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega16 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Block Diagram

Figure 2. Block Diagram



DC Characteristics (Continued)

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (Unless Otherwise Noted)

Symbol	Parameter	Condition	Min	Typ	Max	Units
I_{CC}	Power Supply Current	Active 1 MHz V_{CC} = 3V (ATmega16L)		1.1		mA
		Active 4 MHz, V_{CC} = 3V (ATmega16L)		3.8	5	mA
		Active 8 MHz, V_{CC} = 5V (ATmega16)		12	15	mA
		Idle 1 MHz V_{CC} = 3V (ATmega16L)		0.35		mA
		Idle 4 MHz, V_{CC} = 3V (ATmega16L)		1.2	2	mA
		Idle 8 MHz, V_{CC} = 5V (ATmega16)		5.5	8	mA
	Power-down Mode(5)	WDT enabled, V_{CC} = 3V		< 25	40	µA
		WDT disabled, V_{CC} = 3V		< 10	15	µA
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2			40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Notes:
1. "Max" means the highest value where the pin is guaranteed to be read as low
2. "Min" means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20 mA at Vcc = 5V, 10 mA at Vcc = 3V) under steady state conditions (non-transient), the following must be observed:
   PDIP Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for port A0 - A7, should not exceed 200 mA.
   3] The sum of all IOL, for ports B0 - B7,C0 - C7, D0 - D7 and XTAL2, should not exceed 300 mA.
   TQFP and MLF Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for ports A0 - A7, should not exceed 200 mA.
   3] The sum of all IOL, for ports B0 - B4, should not exceed 200 mA.
   4] The sum of all IOL, for ports B3 - B7, XTAL2, D0 - D2, should not exceed 200 mA.
   5] The sum of all IOL, for ports D3 - D7, should not exceed 200 mA.
   6] The sum of all IOL, for ports C0 - C7, should not exceed 200 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (20 mA at Vcc = 5V, 10 mA at Vcc = 3V) under steady state conditions (non-transient), the following must be observed:
   PDIP Package:
   1] The sum of all IOH, for all ports, should not exceed 400 mA.
   2] The sum of all IOH, for port A0 - A7, should not exceed 200 mA.
   3] The sum of all IOH, for ports B0 - B7,C0 - C7, D0 - D7 and XTAL2, should not exceed 300 mA.
   TQFP and MLF Package:
   1] The sum of all IOH, for all ports, should not exceed 400 mA.
   2] The sum of all IOH, for ports A0 - A7, should not exceed 200 mA.
   3] The sum of all IOH, for ports B0 - B4, should not exceed 200 mA.

SGS-THOMSON MICROELECTRONICS

L293D
L293DD

PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES

- 600mA OUTPUT CURRENT CAPABILITY PER CHANNEL
- 1.2A PEAK OUTPUT CURRENT (non repetitive) PER CHANNEL
- ENABLE FACILITY
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)
- INTERNAL CLAMP DIODES

SO(12+4+4) Powerdip (12+2+2)

ORDERING NUMBERS:

L293DD L293D

DESCRIPTION

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoides, DC and stepping motors) and switching power transistors.

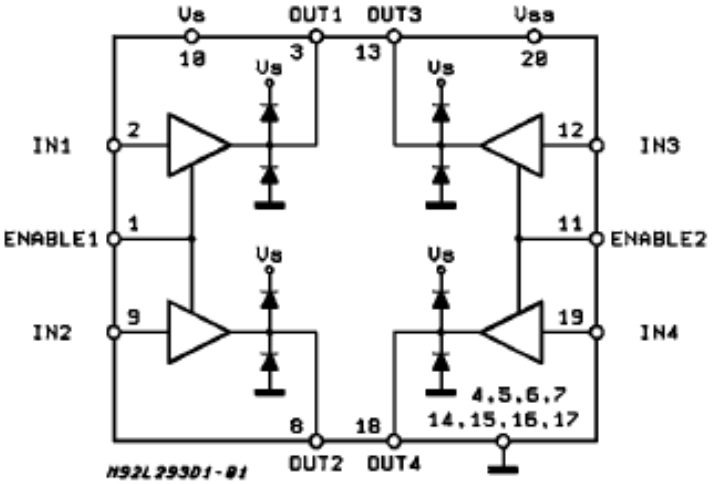
To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

This device is suitable for use in switching applications at frequencies up to 5 kHz.

The L293D is assembled in a 16 lead plastic packaage which has 4 center pins connected together and used for heatsinking

The L293DD is assembled in a 20 lead surface mount which has 8 center pins connected together and used for heatsinking.

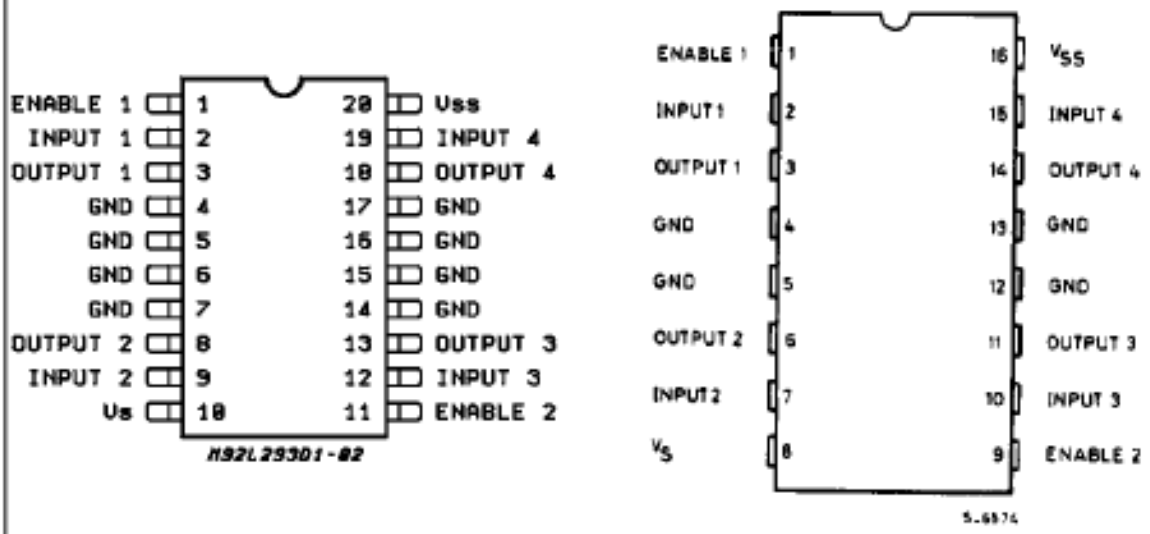
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Supply Voltage	36	V
V_{SS}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{en}	Enable Voltage	7	V
I_o	Peak Output Current (100 µs non repetitive)	1.2	A
P_{tot}	Total Power Dissipation at T_{pins} = 90 °C	4	W
T_{stg}, T_j	Storage and Junction Temperature	– 40 to 150	°C

PIN CONNECTIONS (Top view)



SO(12+4+4) Powerdip(12+2+2)

THERMAL DATA

Symbol	Decription		DIP	SO	Unit
$R_{th\ j\text{-}pins}$	Thermal Resistance Junction-pins	max.	–	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance junction-ambient	max.	80	50 (*)	°C/W
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	max.	14	–	

(*) With 6sq. cm on board heatsink.

ELECTRICAL CHARACTERISTICS (for each channel, V_S = 24 V, V_{SS} = 5 V, T_{amb} = 25 °C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 10)		V_{SS}		36	V
V_{SS}	Logic Supply Voltage (pin 20)		4.5		36	V
I_S	Total Quiescent Supply Current (pin 10)	V_i = L ; I_O = 0 ; V_{en} = H		2	6	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	24	mA
		V_{en} = L			4	mA
I_{SS}	Total Quiescent Logic Supply Current (pin 20)	V_i = L ; I_O = 0 ; V_{en} = H		44	60	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	22	mA
		V_{en} = L		16	24	mA
V_{iL}	Input Low Voltage (pin 2, 9, 12, 19)		– 0.3		1.5	V
V_{iH}	Input High Voltage (pin 2, 9, 12, 19)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
I_{iL}	Low Voltage Input Current (pin 2, 9, 12, 19)	V_{iL} = 1.5 V			– 10	µA
I_{iH}	High Voltage Input Current (pin 2, 9, 12, 19)	2.3 V $\leq V_{iH} \leq V_{SS}$ – 0.6 V		30	100	µA
$V_{en\,L}$	Enable Low Voltage (pin 1, 11)		– 0.3		1.5	V
$V_{en\,H}$	Enable High Voltage (pin 1, 11)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
$I_{en\,L}$	Low Voltage Enable Current (pin 1, 11)	$V_{en\,L}$ = 1.5 V		– 30	– 100	µA
$I_{en\,H}$	High Voltage Enable Current (pin 1, 11)	2.3 V $\leq V_{en\,H} \leq V_{SS}$ – 0.6 V			± 10	µA
$V_{CE(sat)H}$	Source Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = – 0.6 A		1.4	1.8	V
$V_{CE(sat)L}$	Sink Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = + 0.6 A		1.2	1.8	V
V_F	Clamp Diode Forward Voltage	I_O = 600nA		1.3		V
t_r	Rise Time (*)	0.1 to 0.9 V_O		250		ns
t_f	Fall Time (*)	0.9 to 0.1 V_O		250		ns
t_{on}	Turn-on Delay (*)	0.5 V_i to 0.5 V_O		750		ns
t_{off}	Turn-off Delay (*)	0.5 V_i to 0.5 V_O		200		ns

(*) See fig. 1.