Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2020-2021

ΑΘΗΝΑ 16 Νοεμβρίου 2020

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"
2η Εργ. Άσκ. στον Μικροελεγκτή AVR – Χρονιστές, Περιφερειακά (LCD, keyboard), ADC
(υλοποίηση στο εκπαιδευτικό σύστημα easyAVR6**)**

Εξέταση – Επίδειξη: Τετάρτη 25/11/2020.
Προθεσμία για παράδοση Έκθεσης: Κυριακή 30/11/2020

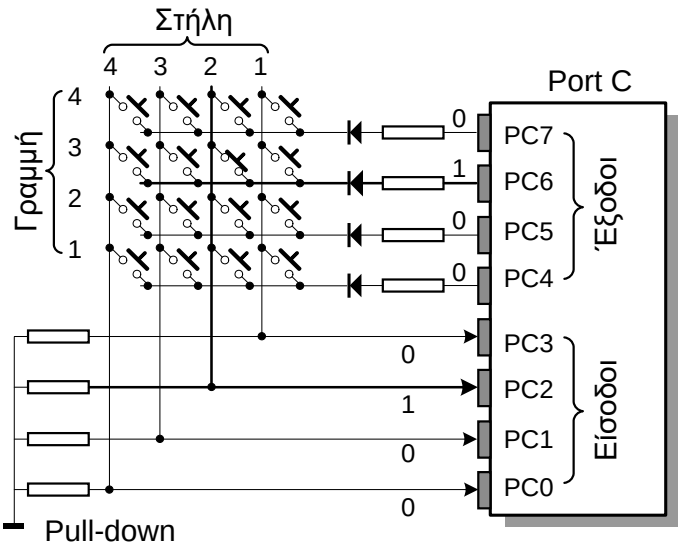
Έλεγχος Περιφερειακών Συσκευών

Η επικοινωνία ενός Μικροελεγκτή με περιφερειακές συσκευές μπορεί να γίνει με διαφορετικούς τρόπους και πρωτόκολλα. Για κάθε διαφορετική περίπτωση και τεχνική επικοινωνίας, είναι ιδιαίτερα χρήσιμη η κατασκευή μιας βιβλιοθήκης λογισμικού (ρουτίνες) που αναλαμβάνουν όλες τις χαμηλού επιπέδου λειτουργίες (π.χ. τροφοδοσία και έλεγχος συγκεκριμένων ακροδεκτών, χρονισμός) που καθορίζονται από τον τρόπο επικοινωνίας, και παρέχουν υψηλότερου επιπέδου λειτουργίες (π.χ. αποστολή και λήψη χαρακτήρων), που χρησιμοποιούνται για την κατασκευή λογισμικού εφαρμογών. Οι βιβλιοθήκες αυτές ονομάζονται οδηγοί συσκευών. Για την κατασκευή τους είναι απαραίτητη η γνώση του τρόπου επικοινωνίας και της συνδεσμολογίας της περιφερειακής συσκευής. Συχνά, η περιφερειακή συσκευή συνοδεύεται από έναν ελεγκτή ο οποίος καθορίζει και υλοποιεί τον τρόπο επικοινωνίας. Τα τεχνικά του χαρακτηριστικά περιέχονται στο αντίστοιχο εγχειρίδιο τεχνικών προδιαγραφών. Η συνδεσμολογία προκύπτει από τα τεχνικά χαρακτηριστικά αλλά και τους περιορισμούς του συστήματος που θα συνδεθεί η περιφερειακή συσκευή (π.χ. διαθεσιμότητα ακροδεκτών).

Η αναπτυξιακή πλακέτα EasyAVR6 διαθέτει μεταξύ των άλλων περιφερειακών συσκευών ένα πληκτρολόγιο 4×4 και μια οθόνη χαρακτήρων 2×16. Στην συνέχεια παρουσιάζονται οι συσκευές αυτές σε συνδυασμό με το αντίστοιχο λογισμικό οδήγησής τους.

Πληκτρολόγιο 4×4

Το πληκτρολόγιο 4×4 του αναπτυξιακού easyAVR6 αποτελείται από 16 πιεστικούς διακόπτες συνδεδεμένους όπως φαίνεται στο ακόλουθο σχήμα:



Σχήμα 3.1 Σχηματικό διάγραμμα διασύνδεσης πληκτρολογίου 4×4 με την θύρα Port C

Η ανάγνωση του πληκτρολογίου γίνεται ανά γραμμή. Η γραμμή που επιθυμούμε να ελέγξουμε για πατημένους διακόπτες επιλέγεται θέτοντας έναν από τους ακροδέκτες PC7 – PC4 της θύρας PORTC του Μικροελεγκτή στο

λογικό 1. Οι 4 γραμμές του πληκτρολογίου επιλέγονται από τους ακροδέκτες PC7 – PC4 οι οποίοι πρέπει να είναι ρυθμισμένοι για έξοδο. Στη συνέχεια εντοπίζουμε τις στήλες των πιεσμένων διακοπτών, διαβάζοντας τον καταχωρητή PINC και ελέγχοντας τους ακροδέκτες PC3 – PC0 που τους έχουμε ρυθμίσει ως εισόδους. Λογικό 1 αντιστοιχεί σε πιεσμένο διακόπτη. Αν δεν υπάρχει πιεσμένος διακόπτης, λόγω των pull-down αντιστάσεων, έχουμε λογικό 0 στην είσοδο.

Η αρχικοποίηση της θύρα PORTC του Μικροελεγκτή μπορεί να πραγματοποιηθεί με τις ακόλουθες εντολές (οι εσωτερικές αντιστάσεις pull-up πρέπει να είναι απενεργοποιημένες).

```
ldi r24, (1 << PC7) | (1 << PC6) | (1 << PC5) | (1 << PC4)        ; θέτει ως εξόδους τα 4 MSB
out DDRC, r24                                                     ; της θύρας PORTC
```

Για να ελέγξουμε μια γραμμή του πληκτρολογίου για πιεσμένους διακόπτες μπορούμε να χρησιμοποιήσουμε τη ρουτίνα scan_row. Στον καταχωρητή r24 αποθηκεύουμε τον αριθμό της γραμμής (1 – 4) που θέλουμε να ελέγξουμε και ανακτούμε το αποτέλεσμα του ελέγχου στα 4 λιγότερα σημαντικά bit του ίδιου καταχωρητή. Οι δύο εντολές *nop* μεταξύ της εντολής που επιλέγει τη γραμμή και της εντολής που διαβάζει τον καταχωρητή PINC είναι απαραίτητες καθώς μπορεί να απαιτηθούν μέχρι και δύο κύκλοι ρολογιού μέχρι μια αλλαγή στην κατάσταση των ακροδεκτών (λόγω της αδράνειας τους) να καταγραφεί στον καταχωρητή PINC.

Ρουτίνα: scan_row_*sim*
Έλεγχος μιας γραμμής του πληκτρολογίου για πιεσμένους διακόπτες.
Είσοδος: Ο αριθμός της γραμμής προς ανάγνωση πρέπει να είναι αποθηκευμένος στον καταχωρητή r24 (τιμή 1-4).
Έξοδος: Στα 4 λιγότερο σημαντικά bit του r24 είναι αποθηκευμένη η κατάσταση κάθε διακόπτη.
Καταχωρητές: r25: r24
Καλούμενες υπορουτίνες: -

```
scan_row_sim:
out PORTC, r25          ; η αντίστοιχη γραμμή τίθεται στο λογικό '1'

push r24                ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25                ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24,low(500)        ; πρόσβασης
ldi r25,high(500)
rcall wait_usec
pop r25
pop r24                 ; τέλος τμήμα κώδικα

nop
nop                     ; καθυστέρηση για να προλάβει να γίνει η αλλαγή κατάστασης
in r24, PINC            ; επιστρέφουν οι θέσεις (στήλες) των διακοπτών που είναι πιεσμένοι
andi r24 ,0x0f          ; απομονώνονται τα 4 LSB όπου τα '1' δείχνουν που είναι πατημένοι
ret                     ; οι διακόπτες.
```

Τώρα που διαθέτουμε μια ρουτίνα που ελέγχει μια γραμμή του πληκτρολογίου για πιεσμένους διακόπτες μπορούμε να γράψουμε μια άλλη που θα την ενσωματώνει και θα ελέγχει ολόκληρο το πληκτρολόγιο. Το αποτέλεσμα του ελέγχου είναι ένας δυαδικός αριθμός με μήκος 16 bit που αποθηκεύεται στους καταχωρητές r25: r24. Κάθε διακόπτης από τους 16 αντιστοιχεί σε ένα bit των καταχωρητών r25: r24 όπου λογικό '1' σημαίνει ότι είναι πιεσμένος ο αντίστοιχος διακόπτης. Η αντιστοιχία φαίνεται στο επόμενο σχήμα.

r25								r24							
A	3	2	1	B	6	5	4	C	9	8	7	D	#	0	*

Ρουτίνα: scan_keypad_*sim*
Έλεγχος του πληκτρολογίου για πιεσμένους διακόπτες.
Είσοδος: καμία
Έξοδος: Στους καταχωρητές r25: r24 είναι αποθηκευμένη η κατάσταση των 16 διακοπτών του πληκτρολογίου.
Καταχωρητές: r27: r26, r25: r24
Καλούμενες υπορουτίνες: scan_row_*sim*

```
scan_keypad_sim:
push r26                ; αποθήκευσε τους καταχωρητές r27:r26 γιατι τους
push r27                ; αλλάζουμε μέσα στην ρουτίνα
ldi r25 , 0x10          ; έλεγξε την πρώτη γραμμή του πληκτρολογίου (PC4: 1 2 3 A)
rcall scan_row_sim
swap r24                ; αποθήκευσε το αποτέλεσμα
mov r27, r24            ; στα 4 msb του r27
ldi r25 ,0x20           ; έλεγξε τη δεύτερη γραμμή του πληκτρολογίου (PC5: 4 5 6 B)
rcall scan_row_sim
add r27, r24            ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r27
ldi r25 , 0x40          ; έλεγξε την τρίτη γραμμή του πληκτρολογίου (PC6: 7 8 9 C)
rcall scan_row_sim
swap r24                ; αποθήκευσε το αποτέλεσμα
mov r26, r24            ; στα 4 msb του r26
ldi r25 ,0x80           ; έλεγξε την τέταρτη γραμμή του πληκτρολογίου (PC7: * 0 # D)
rcall scan_row_sim
add r26, r24            ; αποθήκευσε το αποτέλεσμα στα 4 lsb του r26
movw r24, r26           ; μετέφερε το αποτέλεσμα στους καταχωρητές r25:r24

clr r26                 ; προστέθηκε για την απομακρυσμένη πρόσβαση
out PORTC,r26           ; προστέθηκε για την απομακρυσμένη πρόσβαση

pop r27                 ; επανάφερε τους καταχωρητές r27:r26
pop r26
ret
```

Έως τώρα έχουμε δει πώς να αρχικοποιούμε την θύρα PORTC του Μικροελεγκτή για να χρησιμοποιήσουμε το πληκτρολόγιο και πώς να το ελέγξουμε για πιεσμένα πλήκτρα. Όμως αυτό που πραγματικά μας ενδιαφέρει δεν είναι ποιοι διακόπτες είναι πιεσμένοι τη στιγμή που ελέγχουμε το πληκτρολόγιο, αλλά ποιο πλήκτρο πάτησε ο χρήστης. Όταν ο χρήστης πατάει ένα πλήκτρο, αυτό μπορεί να μείνει πιεσμένο για αυθαίρετα μεγάλο χρονικό διάστημα. **Πρέπει να μπορούμε να ξεχωρίσουμε ποιο πλήκτρο πατήθηκε (ως ολοκληρωμένη ενέργεια) και όχι ποιο πλήκτρο είναι πατημένο.**

Για να το επιτύχουμε πρέπει να ελέγχουμε το πληκτρολόγιο για πιεσμένους διακόπτες, με την ρουτίνα scan_keypad, **να αποθηκεύουμε το αποτέλεσμα στη μνήμη RAM του Μικροελεγκτή** και στη συνέχεια με μια δεύτερη κλήση της ίδιας ρουτίνας να ξαναελέγξουμε το πληκτρολόγιο. Μια σύγκριση των δύο καταστάσεων του πληκτρολογίου θα αποκαλύψει τις διαφορές στην κατάσταση των διακοπτών. Το χρονικό διάστημα ανάμεσα στις διαδοχικές κλήσεις της συνάρτησης *scan_keypad* είναι κρίσιμο, διότι καθορίζει το χρόνο που θα πρέπει να μείνει πιεσμένος ένας διακόπτης από το χρήστη για να καταγραφεί από τον Μικροελεγκτή. Αυτό σημαίνει ότι ένα μεγάλο χρονικό διάστημα μεταξύ των διαδοχικών κλήσεων της ρουτίνας θα αναγκάσει τον χρήστη να κρατά πατημένο το πλήκτρο για αντίστοιχα μεγάλο χρονικό διάστημα (ώστε να μπορεί να αναγνωριστεί). Αντίθετα, ένα πολύ μικρό χρονικό διάστημα θα δημιουργήσει προβλήματα λόγω του σπινθηρισμού που παρουσιάζουν οι διακόπτες.

Η ρουτίνα *scan_keypad_rising_edge_sim* υλοποιεί όσα αναφέρθηκαν πρωτύτερα και παράλληλα αντιμετωπίζει αποτελεσματικά το ζήτημα του σπινθηρισμού των διακοπτών.

Ρουτίνα: scan_keypad_rising_edge_*sim*
Έλεγχος του πληκτρολογίου για διακόπτες που δεν ήταν πιεσμένοι την τελευταία φορά που κλήθηκε η ρουτίνα και τώρα είναι.
Είσοδος: Ο αναμενόμενος χρόνος σπινθηρισμού των διακοπτών σε ms είναι αποθηκευμένος στον καταχωρητή r24.
Έξοδος: Ένας αριθμός των 16 bit, ενδεικτικός των διακοπτών που «μόλις» πατήθηκαν, είναι αποθηκευμένος στους καταχωρητές r25: r24
Καταχωρητές: r27: r26, r25: r24, r23: r22
Καλούμενες υπορουτίνες: scan_keypad_*sim*, wait_msec

Παρατηρήσεις:

Για να καταγραφούν οι διακόπτες που έχουν «μόλις» πιεστεί, η ρουτίνα χρησιμοποιεί την μεταβλητή _tmp_. Επειδή η μεταβλητή _tmp_ βρίσκεται στην RAM του Μικροελεγκτή δεν είναι αρχικοποιημένη. Ο χρήστης πρέπει να την αρχικοποιήσει είτε ρητά είτε με μια κλήση της ρουτίνας μόνο για αυτό το σκοπό.

```
        ; ---- Αρχή τμήματος δεδομένων

.DSEG
_tmp_: .byte 2

        ; ---- Τέλος τμήματος δεδομένων

.CSEG
```

```
scan_keypad_rising_edge_sim:
push r22                ; αποθήκευσε τους καταχωρητές r23:r22 και τους
push r23                ; r26:r27 γιατι τους αλλάζουμε μέσα στην ρουτίνα
push r26
push r27
rcall scan_keypad_sim   ; έλεγξε το πληκτρολόγιο για πιεσμένους διακόπτες
push r24                ; και αποθήκευσε το αποτέλεσμα
push r25
ldi r24 ,15             ; καθυστέρησε 15 ms (τυπικές τιμές 10-20 msec που καθορίζεται από τον
ldi r25 ,0              ; κατασκευαστή του πληκτρολογίου – χρονοδιάρκεια σπινθηρισμών)
rcall wait_msec
rcall scan_keypad_sim   ; έλεγξε το πληκτρολόγιο ξανά και απόρριψε
pop r23                 ; όσα πλήκτρα εμφανίζουν σπινθηρισμό
pop r22
and r24 ,r22
and r25 ,r23
ldi r26 ,low(_tmp_)     ; φόρτωσε την κατάσταση των διακοπτών στην
ldi r27 ,high(_tmp_)    ; προηγούμενη κλήση της ρουτίνας στους r27:r26
ld r23 ,X+
ld r22 ,X
st X ,r24               ; αποθήκευσε στη RAM τη νέα κατάσταση
st -X ,r25              ; των διακοπτών
com r23
com r22                 ; βρες τους διακόπτες που έχουν «μόλις» πατηθεί
and r24 ,r22
and r25 ,r23

pop r27                 ; επανάφερε τους καταχωρητές r27:r26
pop r26                 ; και r23:r22
pop r23
pop r22
ret
```

Τέλος, επειδή η κατάσταση του πληκτρολογίου στην μορφή ενός δυαδικού αριθμού δεν είναι ιδιαίτερα χρήσιμη υπάρχει και η ρουτίνα *keypad_to_ascii* που εντοπίζει τον διακόπτη που έχει πατηθεί και επιστρέφει τον κωδικό ascii του χαρακτήρα που αντιστοιχεί στον διακόπτη. Αν δεν είναι πιεσμένος κανένας διακόπτης επιστρέφει την τιμή 0, ενώ εάν είναι πατημένοι πολλοί επιστρέφει μόνο έναν από αυτούς (ο $1^{ος}$ που εντοπίζεται με βάση την σειρά εξερεύνησης των εντολών της ρουτίνας που ακολουθεί).

Ρουτίνα: keypad_to_ascii_*sim*
Αντιστοίχιση διακοπτών, κωδικών ascii.
Είσοδος: Στους καταχωρητές r25: r24 είναι αποθηκευμένος ένας αριθμός 16 bit, ενδεικτικός της κατάστασης κάθε διακόπτη.
Έξοδος: Ο κωδικός ascii, που αντιστοιχεί στον πρώτο πατημένο διακόπτη που εντοπίστηκε, αποθηκεύεται στον καταχωρητή r24 ή 0 αν δεν έχει πατηθεί κάποιος.
Καταχωρητές: r27: r26, r25: r24

Παρατηρήσεις: Ο αριθμός των 16 bit που αποθηκεύεται στους καταχωρητές r25:r24 κατά την κλήση της ρουτίνας πρέπει να προέρχεται από μια εκ των *scan_keypad_sim* ή *scan_keypad_rising_edge_sim*.

```
keypad_to_ascii_sim:
push r26                ; αποθήκευσε τους καταχωρητές r27:r26 γιατι τους
push r27                ; αλλάζουμε μέσα στη ρουτίνα
movw r26 ,r24           ; λογικό '1' στις θέσεις του καταχωρητή r26 δηλώνουν
                        ; τα παρακάτω σύμβολα και αριθμούς
ldi r24 ,'*'
```

; r26							
;C	9	8	7	D	#	0	*

```
sbrc r26 ,0
rjmp return_ascii
ldi r24 ,'0'
sbrc r26 ,1
rjmp return_ascii
ldi r24 ,'#'
sbrc r26 ,2
rjmp return_ascii
ldi r24 ,'D'
sbrc r26 ,3             ; αν δεν είναι '1'παρακάμπτει την ret, αλλιώς (αν είναι '1')
rjmp return_ascii       ; επιστρέφει με τον καταχωρητή r24 την ASCII τιμή του D.
ldi r24 ,'7'
sbrc r26 ,4
rjmp return_ascii
ldi r24 ,'8'
sbrc r26 ,5
rjmp return_ascii
ldi r24 ,'9'
sbrc r26 ,6
rjmp return_ascii       ;
ldi r24 ,'C'
sbrc r26 ,7
rjmp return_ascii
ldi r24 ,'4'            ; λογικό '1' στις θέσεις του καταχωρητή r27 δηλώνουν
sbrc r27 ,0             ; τα παρακάτω σύμβολα και αριθμούς
rjmp return_ascii
ldi r24 ,'5'
```

;r27							
;A	3	2	1	B	6	5	4

```
sbrc r27 ,1
rjmp return_ascii
ldi r24 ,'6'
sbrc r27 ,2
rjmp return_ascii
ldi r24 ,'B'
sbrc r27 ,3
rjmp return_ascii
ldi r24 ,'1'
sbrc r27 ,4
rjmp return_ascii       ;
ldi r24 ,'2'
sbrc r27 ,5
rjmp return_ascii
ldi r24 ,'3'
```

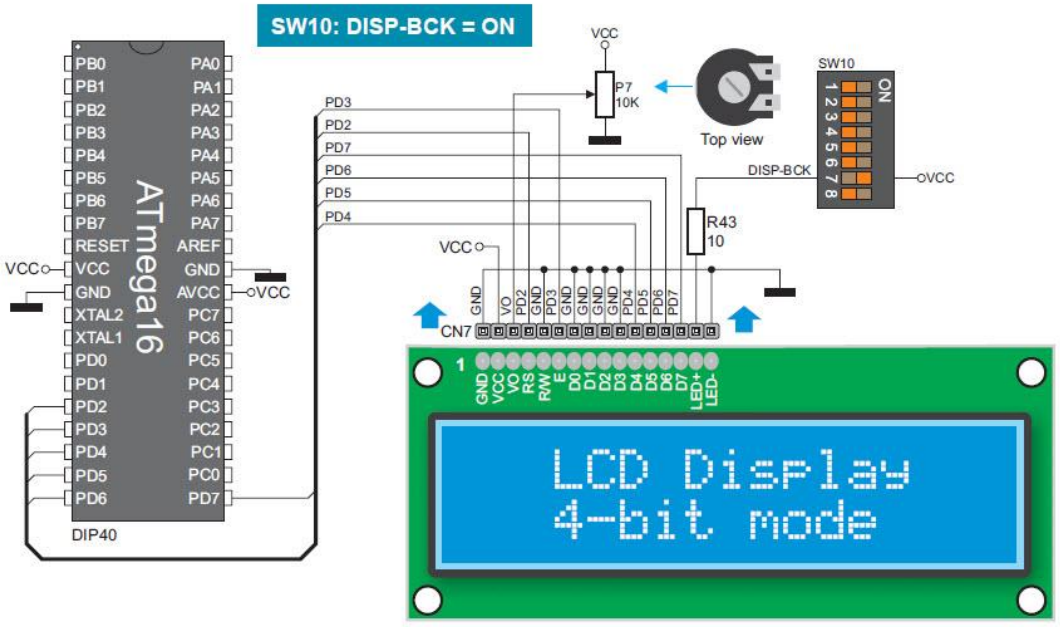
```
sbrc r27 ,6
rjmp return_ascii
ldi r24 ,'A'
sbrc r27 ,7
rjmp return_ascii
clr r24
rjmp return_ascii

return_ascii:
pop r27                    ; επανάφερε τους καταχωρητές r27:r26
pop r26
ret
```

Χρήση αλφαριθμητικής οθόνη LCD

Επίσης στην άσκηση αυτή θα γίνει μελέτη χρήσης της αλφαριθμητικής οθόνη LCD 2×16 χαρακτήρων (επικοινωνία μεταξύ οθόνης και του μικροελεγκτή γίνεται με λέξεις των 4 bit). Στην συνέχεια παρουσιάζεται η συσκευή αυτή σε συνδυασμό με το αντίστοιχο λογισμικό οδήγησής της.

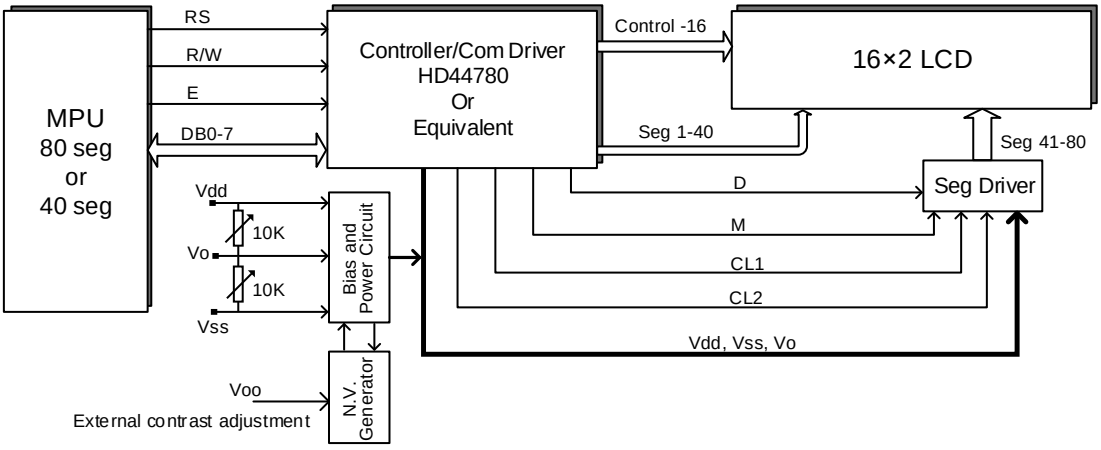
Αλφαριθμητική Οθόνη Χαρακτήρων 2×16

Στην αναπτυξιακή πλακέτα EasyAVR6 μια πολύ χρήσιμη περιφερειακή συσκευή είναι η πρόσθετη οθόνη χαρακτήρων 2×16. Η συνδεσμολογία γίνεται με τους 6 περισσότερο σημαντικούς ακροδέκτες της θύρας PORTD (PD2-PD7), όπως φαίνεται στο σχήμα 3.2. Οι ακροδέκτες αυτοί συνδέονται σε ακροδέκτες του ελεγκτή της οθόνης WH1602B, του οποίου το εγχειρίδιο τεχνικών προδιαγραφών παρατίθεται στις επόμενες σελίδες. Από τους 6 ακροδέκτες, οι 2 λιγότερο σημαντικοί (PD2 και PD3) είναι ακροδέκτες ελέγχου ενώ οι υπόλοιποι 4 σχηματίζουν λέξεις των 4 bit με τις οποίες γίνεται η επικοινωνία μικροελεγκτή – ελεγκτή οθόνης.



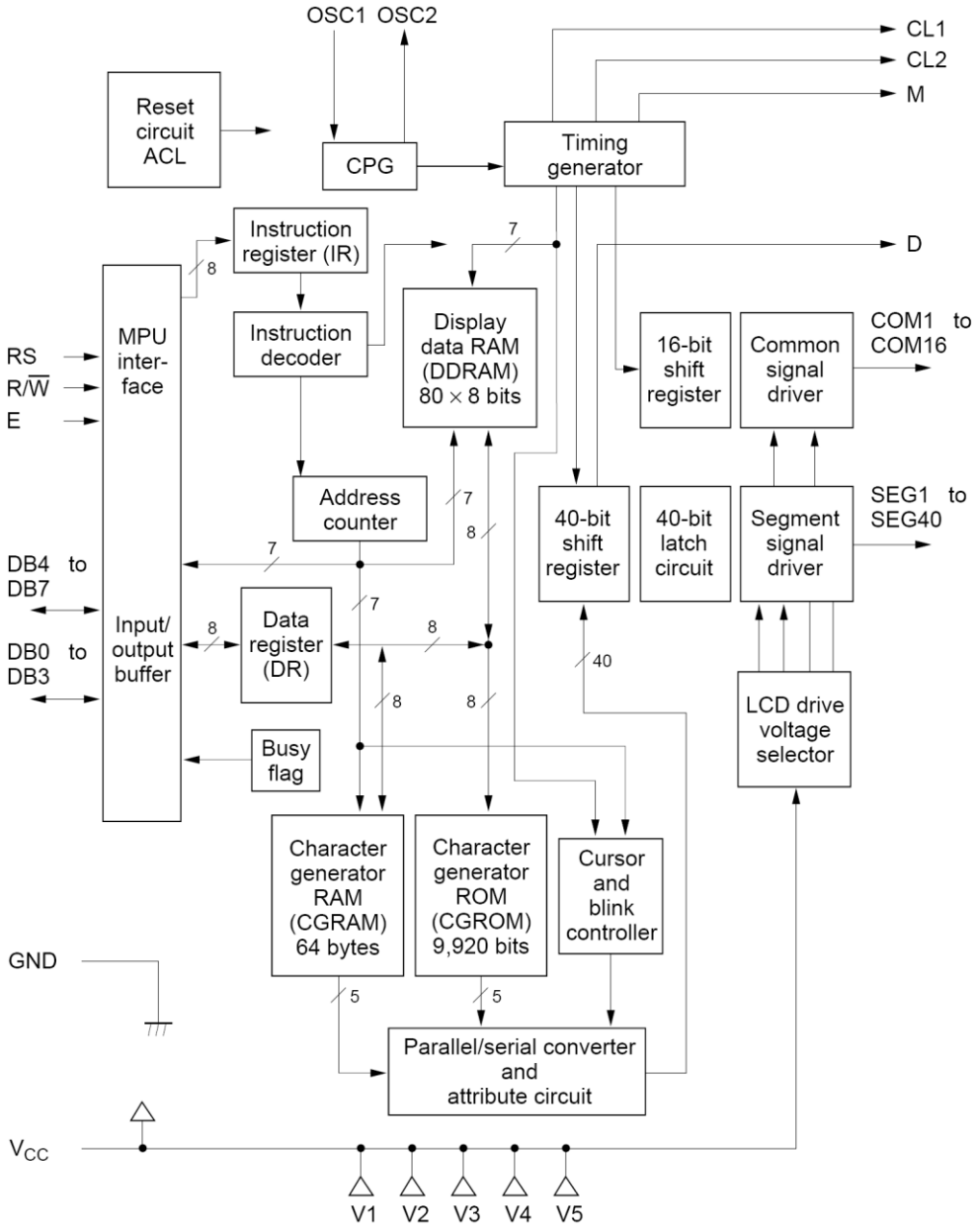
Σχήμα 3.2. Κυκλωματικό διάγραμμα οθόνης χαρακτήρων 2×16.

Για να χρησιμοποιήσουμε την αλφαριθμητική οθόνη χαρακτήρων του easyAVR6 είναι απαραίτητο να εξοικειωθούμε με τη δομή και τη λειτουργία του. Η εσωτερική οργάνωση της αλφαριθμητικής οθόνης χαρακτήρων παρουσιάζεται στο Σχήμα 3.3.



Σχήμα 3.3. Κυκλωματικό διάγραμμα οθόνης χαρακτήρων 2×16.

Κεντρικό ρόλο παίζει ο ελεγκτής HD44780, ο οποίος αναλαμβάνει να παρουσιάσει στην οθόνη υγρών κρυστάλλων τους αλφαριθμητικούς χαρακτήρες καθώς και την αλληλεπίδραση με τον χρήστη. Το μπλοκ διάγραμμα του ελεγκτή HD44780 φαίνεται στο Σχήμα 3.4.



Σχήμα 3.4. Κυκλωματικό διάγραμμα οθόνης χαρακτήρων 2×16.

Περιγραφή λειτουργίας
Η μονάδα οθόνης LCD περιλαμβάνει ελεγκτή που έχει δύο 8-bit καταχωρητές, έναν καταχωρητή εντολών (instruction register -IR) και έναν καταχωρητή δεδομένων (data register- DR).

Καταχωρητές
Ο καταχωρητής IR αποθηκεύει κωδικούς εντολών, όπως καθαρισμός οθόνης ολίσθηση δρομέα, και πληροφορίες για τη διεύθυνση των δεδομένων της RAM απεικόνισης (display data RAM -DDRAM) και για τη γεννήτρια χαρακτήρων RAM (character generator RAM- CGRAM). Ο καταχωρητής IR εγγράφεται μόνο από τον Μικροελεγκτή. Ο καταχωρητής DR αποθηκεύει προσωρινά τα δεδομένα για να εγγράφουν σε DDRAM ή CGRAM και προσωρινά αποθηκεύει δεδομένα που πρέπει να διαβαστούν από την DDRAM ή την CGRAM. Τα δεδομένα γράφονται στον DR από τον Μικροελεγκτή αυτόματα μεταφέρονται σε DDRAM ή CGRAM από μια εσωτερική λειτουργία. Το DR χρησιμοποιείται επίσης για αποθήκευση κατά την ανάγνωση δεδομένων από DDRAM ή CGRAM. Όταν πληροφορίες διεύθυνσης εγγράφονται στο IR, τα αντίστοιχα δεδομένα αποθηκεύονται στον DR από την DDRAM ή την CGRAM μέσω μιας εσωτερικής λειτουργίας. Η μεταφορά δεδομένων στον MPU ολοκληρώνεται όταν διαβάσει τον DR. Μετά την ανάγνωση, τα δεδομένα των DDRAM ή CGRAM της επόμενης διεύθυνσης αποστέλλονται στον DR για την επόμενη ανάγνωση από τον Μικροελεγκτή. Με το σήμα επιλογής καταχωρητή (register selector -RS), οι δύο αυτοί καταχωρητές μπορούν να επιλεγούν (Πίνακας 3.1).

Σημαία απασχόλησης (Busy Flag-BF)
Όταν η σημαία απασχόλησης είναι 1, η HD44780U είναι σε κατάσταση εσωτερικής λειτουργίας, και επόμενη εντολή δεν γίνεται δεκτή. Όταν RS = 0 και R / W = 1 (Πίνακας 1), η σημαία απασχόλησης εμφανίζεται στην έξοδο DB7. Η επόμενη εντολή πρέπει να είναι δοθεί, μετά την εξασφάλιση ότι η σημαία απασχόλησης είναι 0.

Μετρητής Διεύθυνση (Address Counter -AC)
Ο μετρητής διεύθυνση (AC) παρέχει διευθύνσεις σε αμφότερες τις μνήμες DDRAM και CGRAM. Όταν μια διεύθυνση μιας εντολής εγγράφεται στον IR, η διεύθυνση στέλνεται από τον IR στον AC. Η επιλογή είτε της DDRAM ή της CGRAM καθορίζεται επίσης από την εντολή.
Μετά την εγγραφή προς (ή την ανάγνωση από) τις DDRAM ή CGRAM, η AC αυτόματα αυξάνεται κατά 1 (ή ελαττώνεται κατά 1). Το περιεχόμενο του καταχωρητή AC δίνεται στη έξοδο μέσω των DB0 - DB6 όταν RS = 0 και R / W = 1 (βλέπε Πίνακα 3.1).

Πίνακας 3.1. Επιλογή Καταχωρητή

RS	R/W	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

Περιγραφή των εσωτερικών δομικών μονάδων του ελεγκτή

Μνήμες
Ο ελεγκτής διαθέτει συνολικά 3 μνήμες, δύο τύπου RAM και μια τύπου ROM. Η ROM δημιουργίας χαρακτήρων χρησιμοποιείται για τη δημιουργία εικόνων χαρακτήρων 5×8 κουκίδων ή 5×10 κουκίδων από κωδικούς χαρακτήρων των 8 bit. Η αντιστοιχία αριθμών των 8 bit και χαρακτήρων φαίνεται στον ακόλουθο πίνακα.
Η RAM δημιουργίας χαρακτήρων (character generator RAM- CGRAM) εκτελεί την ίδια λειτουργία με την ROM δημιουργίας χαρακτήρων, με την διαφορά ότι οι χαρακτήρες που απεικονίζονται πρέπει να δημιουργηθούν από τον χρήστη και να αποθηκευτούν σε αυτή.

Πίνακας 3.2. Αντιστοιχία αριθμών των 8 bit και χαρακτήρων.

Upper 4 bit / Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	CG RAM (2)															
LLHL	CG RAM (3)															
LLHH	CG RAM (4)															
LHLL	CG RAM (5)															
LHLH	CG RAM (6)															
LHHL	CG RAM (7)															
LHHH	CG RAM (8)															
HLLL	CG RAM (1)															
HLLH	CG RAM (2)															
HLHL	CG RAM (3)															
HLHH	CG RAM (4)															
HHLL	CG RAM (5)															
HHLH	CG RAM (6)															
HHHL	CG RAM (7)															
HHHH	CG RAM (8)															

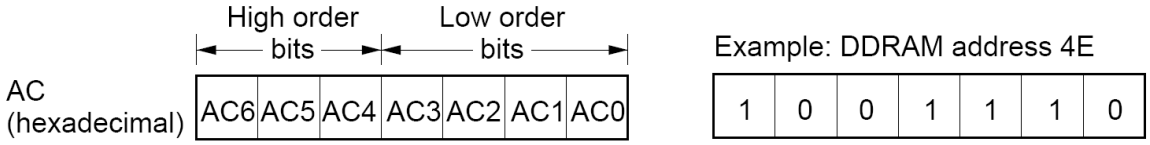
Η RAM απεικόνισης δεδομένων (display data RAM –**DDRAM)** αποθηκεύει τα δεδομένα που πρόκειται να απεικονιστούν στην οθόνη. Η χωρητικότητά της είναι 80 byte. Η αντιστοιχία μεταξύ θέσεων στην RAM απεικόνισης χαρακτήρων και θέσεων στην οθόνη φαίνεται στην ακόλουθη εικόνα. Παρατηρήστε ότι αρκετές θέσεις της μνήμης δεν απεικονίζονται στην οθόνη και πως όταν η οθόνη έχει διαμόρφωση για απεικόνιση χαρακτήρων σε δύο γραμμές, οι διευθύνσεις που απεικονίζονται στην πρώτη και δεύτερη γραμμή της οθόνης δεν είναι συνεχόμενες.

DDRAM Address

Display Data RAM (DDRAM)

Display data RAM (DDRAM) stores display data represented in 8-bit character codes. Its extended capacity is 80 × 8 bits, or 80 characters. The area in display data RAM (DDRAM) that is not used for display can be used as general data RAM. See Figure 1 for the relationships between DDRAM addresses and positions on the liquid crystal display.

The DDRAM address (ADD) is set in the address counter (AC) as hexadecimal.



1-line display (N = 0)

— When there are fewer than 80 display characters, the display begins at the head position. For example, if using only the HD44780, 8 characters are displayed. See Figure 3.

When the display shift operation is performed, the DDRAM address shifts.

Display position (digit)	1	2	3	4	5	···	79	80
DDRAM address (hexadecimal)	00	01	02	03	04	···	4E	4F

2-line display (N = 1)

— Case 1: When the number of display characters is less than 40 × 2 lines, the two lines are displayed from the head. Note that the first line end address and the second line start address are not consecutive. For example, when just the HD44780 is used, 8 characters × 2 lines are displayed.

Display position	1	2	3	4	5	···	39	40
DDRAM address (hexadecimal)	00	01	02	03	04	···	26	27
	40	41	42	43	44	···	66	67

When display shift operation is performed, the DDRAM address shifts.

Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
	40	41	42	43	44	45	46	47
For shift left	01	02	03	04	05	06	07	08
	41	42	43	44	45	46	47	48
For shift right	27	00	01	02	03	04	05	06
	67	40	41	42	43	44	45	46

— Case 2: For a 16-character × 2-line display, the HD44780 can be extended using one 40-output extension driver.

When display shift operation is performed, the DDRAM address shifts.

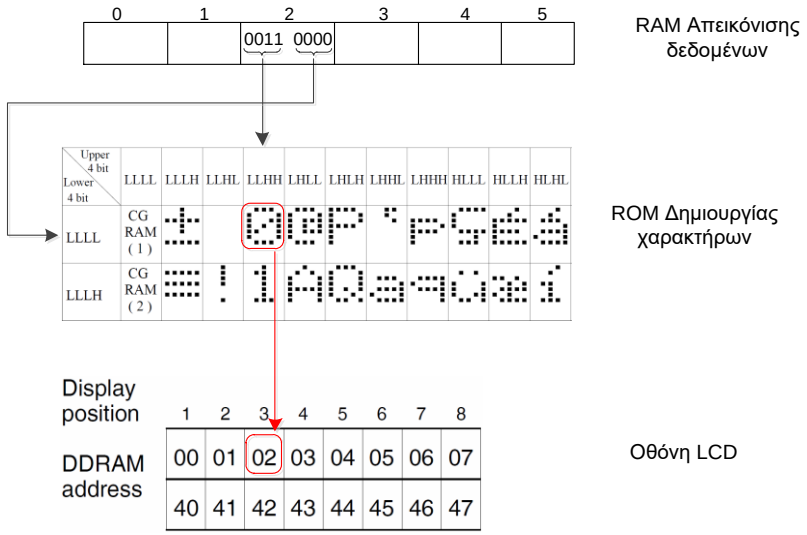
Display position	1	2	3	4	5	6	7	8
DDRAM address	00	01	02	03	04	05	06	07
For shift left	01	02	03	04	05	06	07	08
For shift right	4F	00	01	02	03	04	05	06

1-Line by 8-Character Display Example

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
	HD44780U display								Extension driver display							
For shift left	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50
For shift right	27	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
	67	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E

2-Line by 16-Character Display Example

Ο ρόλος των μνημών RAM και ROM στην απεικόνιση δεδομένων στην οθόνη αποδίδεται από το παρακάτω σχεδιάγραμμα.



Καταχωρητές

Ο ελεγκτής HD44780 έχει δύο καταχωρητές των 8 bit που μπορούν να προσπελαστούν άμεσα από το χρήστη. Τον καταχωρητή εντολών και τον καταχωρητή δεδομένων.

Ο καταχωρητής εντολών αποθηκεύει δυαδικούς κωδικούς εντολών και διευθύνσεις για τις μνήμες RAM του ελεγκτή.

Ο καταχωρητής δεδομένων αποθηκεύει προσωρινά τα δεδομένα που πρόκειται να αποθηκευτούν στις RAM του ελεγκτή ή δεδομένα που διαβάστηκαν από αυτές. Δεδομένα που αποθηκεύονται στον καταχωρητή δεδομένων

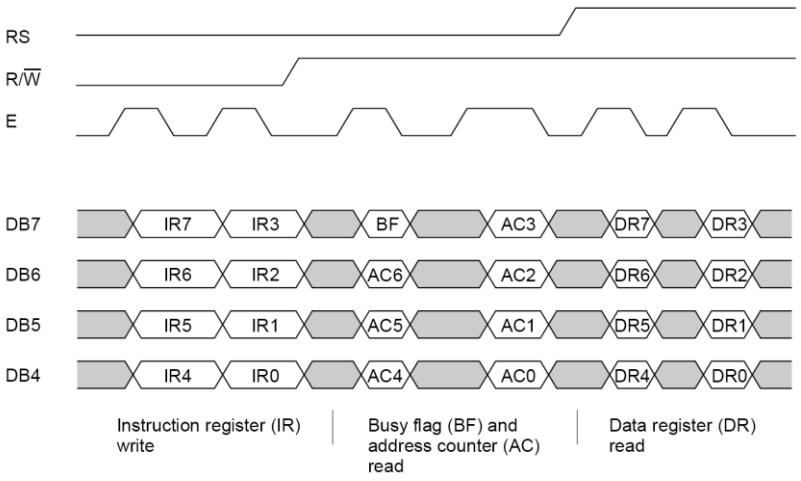
μεταφέρονται αυτόματα στην κατάλληλη RAM. Όταν ο καταχωρητής εντολών δεχτεί πληροφορίες διεύθυνσης, δεδομένα αποθηκεύονται στον καταχωρητή δεδομένων από την κατάλληλη RAM αυτόματα.
Στο μετρητή διευθύνσεων βρίσκεται αποθηκευμένη η τρέχουσα διεύθυνση της RAM απεικόνισης δεδομένων ή της RAM δημιουργίας χαρακτήρων. Το περιεχόμενό του μπορεί να τροποποιηθεί με μια από τις εντολές που αλλάζουν την διεύθυνση κάποιας από τις RAM του ελεγκτή. Το ποια μνήμη επιλέγεται εξαρτάται από την εντολή. Η διεύθυνση που βρίσκεται αποθηκευμένη σε αυτόν τον καταχωρητή αυξάνεται κατά 1(ή μειώνεται κατά 1) αυτόματα με κάθε ανάγνωση/εγγραφή από τη μνήμη.
Η διεπαφή με την οποίαν κάποιος ανταλλάσσει εντολές με τον ελεγκτή φαίνεται στον παρακάτω πίνακα:

Πίνακας 3.3. Η διεπαφή του ελεγκτή HD44780.

Pin No.	Symbol	Level	Description
1	V_{SS}	0V	Ground
2	V_{DD}	5.0V	Supply Voltage for logic
3	VO	(Variable)	Operating voltage for LCD
4	RS	H/L	H: DATA, L: Instruction code
5	R/W	H/L	H: Read(MPU→Module) L: Write(MPU→Module)
6	E	H,H→L	Chip enable signal
7	DB0	H/L	Data bus line
8	DB1	H/L	Data bus line
9	DB2	H/L	Data bus line
10	DB3	H/L	Data bus line
11	DB4	H/L	Data bus line
12	DB5	H/L	Data bus line
13	DB6	H/L	Data bus line
14	DB7	H/L	Data bus line
15	A	—	LED +
16	K	—	LED −

Ο ελεγκτής HD44780 μπορεί να δεχτεί εντολές είτε σε μια ενιαία εντολή των 8 bit είτε σε μια εντολή που αποστέλλεται σε δύο διαδοχικά τμήματα των 4 bit. Στο αναπτυξιακό δεν μπορούμε να στείλουμε εντολές στον ελεγκτή σε μορφή 8 bit, συνεπώς δεν θα ασχοληθούμε άλλο με την μεταφορά εντολών σε αυτή τη μορφή.

Όταν οι εντολές μεταφέρονται σε δύο τμήματα των 4 bit χρησιμοποιούνται μόνο οι γραμμές DB7 – DB4. Τα 4 περισσότερο σημαντικά bit πρέπει να μεταφερθούν πρώτα. Επειδή δεν υπάρχει δυνατότητα ανάγνωσης της σημαίας BUSY FLAG για να γνωρίζουμε πότε ο ελεγκτής είναι έτοιμος να δεχτεί νέα εντολή πρέπει να εισάγουμε καθυστέρηση μεταξύ των διαδοχικών εντολών. Ο χρόνος που χρειάζεται ο ελεγκτής HD44780 για να εκτελέσει κάθε εντολή είναι μεταβλητός και εξαρτάται από την εντολή.



Το σύνολο εντολών του ελεγκτή περιέχεται στον ακόλουθο πίνακα.

Πίνακας 3.4. Οι εντολές του ελεγκτή HD44780.

Instruction	Instruction Code										Description	Execution time (fosc=270Khz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write “00H” to DDRAM and set DDRAM address to “00H” from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	—	Set DDRAM address to “00H” from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39 μs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and blinking of cursor (B) on/off control bit.	39 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	—	—	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39 μs
Function Set	0	0	0	0	1	DL	N	F	—	—	Set interface data length (DL:8-bit/4-bit), numbers of display line (N:2-line/1-line)and, display font type (F:5×11 dots/5×8 dots)	39 μs
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39 μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39 μs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 μs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43 μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43 μs

＊ ”—”：don’t care

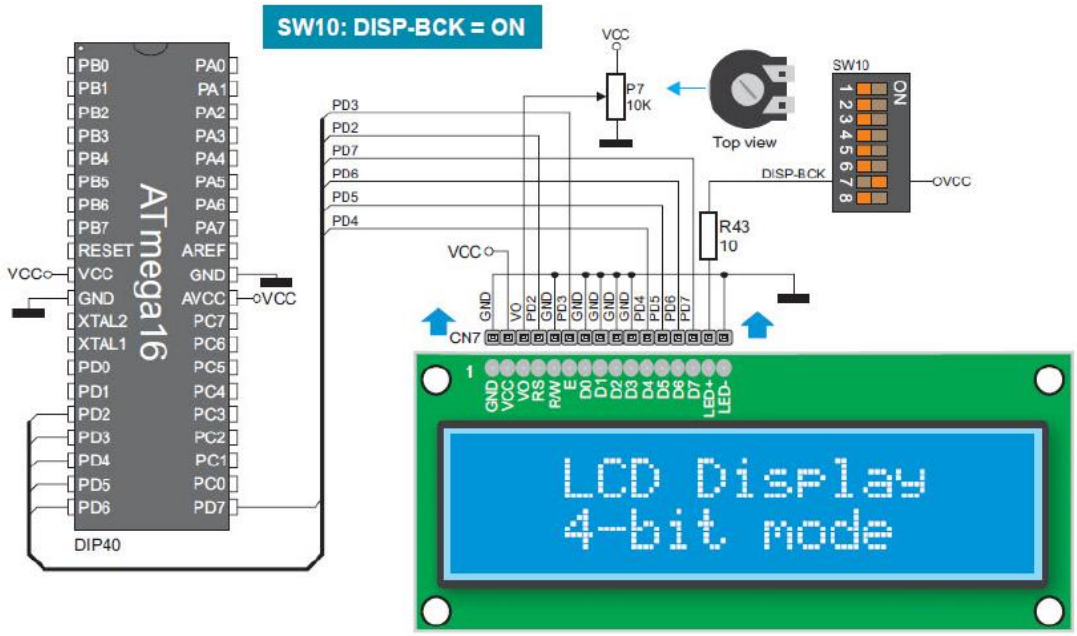
Τέλος για να γίνει δεκτή μια εντολή από τον ελεγκτή πρέπει να τηρούνται οι χρόνοι του παρακάτω πίνακα:

Ta=25℃, VDD=5.0V

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	T_C	1200	—	—	ns
Enable pulse width	T_{PW}	140	—	—	ns
Enable rise/fall time	T_R,T_F	—	—	25	ns
Address set-up time (RS, R/W to E)	t_{AS}	0	—	—	ns
Address hold time	t_{AH}	10	—	—	ns
Data set-up time	t_{DSW}	40	—	—	ns
Data hold time	t_H	10	—	—	ns

Χρήση της οθόνης LCD του αναπτυξιακού easyAVR6.

Η συνδεσμολογία της οθόνης LCD με τον Μικροελεγκτή φαίνεται στο παρακάτω σχήμα (επαναλαμβάνεται για ευκολία το Σχ. 3.2), απ' όπου μπορούμε να συμπεράνουμε ότι οι ακροδέκτες PD7 – PD2 πρέπει να είναι ρυθμισμένοι για έξοδο.



Αρχικά χρειαζόμαστε μια ρουτίνα που θα μεταφέρει τα δύο τμήματα των 4 bit κάθε εντολής. Η ρουτίνα θα πρέπει να αφήνει ανεπηρέαστους τους ακροδέκτες που επιλέγουν μεταξύ καταχωρητή εντολών και καταχωρητή δεδομένων, ώστε να μπορεί να χρησιμοποιηθεί και για τις δύο λειτουργίες.

Ρουτίνα: write_2_nibbles_sim
Αποστολή ενός byte, 4 bit τη φορά στον ελεγκτή της οθόνης LCD. Το λογικό επίπεδο που βρίσκεται ο ακροδέκτης που αντιστοιχεί στο σήμα R/S δεν επηρεάζεται.
Είσοδος: Το byte που μεταδίδεται είναι αποθηκευμένο στον καταχωρητή r24
Έξοδος: -
Καταχωρητές: r25:r24
Καλούμενες υπορουτίνες: -

```
write_2_nibbles_sim:

push r24                ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25                ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24 ,low(6000)      ; πρόσβασης
ldi r25 ,high(6000)
rcall wait_usec
pop r25
pop r24                 ; τέλος τμήμα κώδικα

push r24                ; στέλνει τα 4 MSB
in r25, PIND            ; διαβάζονται τα 4 LSB και τα ξαναστέλνουμε
andi r25, 0x0f          ; για να μην χαλάσουμε την όποια προηγούμενη κατάσταση
andi r24, 0xf0          ; απομονώνονται τα 4 MSB και
add r24, r25            ; συνδυάζονται με τα προϋπάρχοντα 4 LSB
out PORTD, r24          ; και δίνονται στην έξοδο
sbi PORTD, PD3          ; δημιουργείται παλμός Enable στον ακροδέκτη PD3
cbi PORTD, PD3          ; PD3=1 και μετά PD3=0

push r24                ; τμήμα κώδικα που προστίθεται για τη σωστή
```

```
push r25                ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24 ,low(6000)      ; πρόσβασης
ldi r25 ,high(6000)
rcall wait_usec
pop r25
pop r24                 ; τέλος τμήμα κώδικα

pop r24                 ; στέλνει τα 4 LSB. Ανακτάται το byte.
swap r24                ; εναλλάσσονται τα 4 MSB με τα 4 LSB
andi r24 ,0xf0          ; που με την σειρά τους αποστέλλονται
add r24, r25
out PORTD, r24
sbi PORTD, PD3          ; Νέος παλμός Enable
cbi PORTD, PD3
ret
```

Στη συνέχεια, με βάση την προηγούμενη ρουτίνα μπορούμε να δημιουργήσουμε δύο άλλες. Η μία θα στέλνει εντολές στην οθόνη και η άλλη δεδομένα.

Ρουτίνα: lcd_data_sim
Αποστολή ενός byte δεδομένων στον ελεγκτή της οθόνης lcd. Ο ελεγκτής πρέπει να βρίσκεται σε 4 bit mode.
Είσοδος: Το byte που μεταδίδεται είναι αποθηκευμένο στον καταχωρητή r24
Καταχωρητές: r25:r24
Καλούμενες υπορουτίνες: *wait_usec, write_2_nibbles_sim*

```
lcd_data_sim:
push r24                ; αποθήκευσε τους καταχωρητές r25:r24 γιατί τους
push r25                ; αλλάζουμε μέσα στη ρουτίνα
sbi PORTD, PD2          ; επιλογή του καταχωρητή δεδομένων (PD2=1)
rcall write_2_nibbles_sim ; αποστολή του byte
ldi r24 ,43             ; αναμονή 43μsec μέχρι να ολοκληρωθεί η λήψη
ldi r25 ,0              ; των δεδομένων από τον ελεγκτή της lcd
rcall wait_usec
pop r25                 ;επανάφερε τους καταχωρητές r25:r24
pop r24
ret
```

Ρουτίνα: lcd_command_sim
Αποστολή μιας εντολής στον ελεγκτή της οθόνης lcd. Ο ελεγκτής πρέπει να βρίσκεται σε 4 bit mode.
Είσοδος: Η εντολή που μεταδίδεται είναι αποθηκευμένη στον καταχωρητή r24
Έξοδος: -
Καταχωρητές: r25:r24
Καλούμενες υπορουτίνες: wait_usec, write_2_nibbles_*sim*

```
lcd_command_sim:
push r24                ; αποθήκευσε τους καταχωρητές r25:r24 γιατί τους
push r25                ; αλλάζουμε μέσα στη ρουτίνα
cbi PORTD, PD2          ; επιλογή του καταχωρητή εντολών (PD2=0)
rcall write_2_nibbles_sim ; αποστολή της εντολής και αναμονή 39μsec
ldi r24, 39             ; για την ολοκλήρωση της εκτέλεσης της από τον ελεγκτή της lcd.
ldi r25, 0              ; ΣΗΜ.: υπάρχουν δύο εντολές, οι clear display και return home,
rcall wait_usec         ; που απαιτούν σημαντικά μεγαλύτερο χρονικό διάστημα.
pop r25                 ; επανάφερε τους καταχωρητές r25:r24
pop r24
ret
```

Τώρα που μπορούμε να στείλουμε εντολές και δεδομένα στην οθόνη μένει να την αρχικοποιήσουμε στην επιθυμητή κατάσταση, ώστε να μπορεί να χρησιμοποιηθεί. Όταν η οθόνη τροφοδοτείται με ρεύμα για πρώτη φορά ο ελεγκτής

HD44780 πραγματοποιεί μια εσωτερική αρχικοποίηση και για αυτό απαιτείται να περιμένουμε 40 ms. Στη συνέχεια ο ελεγκτής βρίσκεται σε 8 bit mode και είναι έτοιμος να λάβει εντολές. Ο κώδικας που θα κάνει την αρχικοποίηση δεν πρέπει να βασίζεται στο ότι ο ελεγκτής βρίσκεται σε 8 bit mode, διότι αυτό δεν είναι πάντα αληθές. Κάθε φορά που προγραμματίζουμε τον Μικροελεγκτή, αυτός ξεκινάει την εκτέλεση του κώδικα από την αρχή, η οθόνη όμως βρίσκεται στην κατάσταση που την αφήσαμε την προηγούμενη φορά. Για να οδηγήσουμε την οθόνη σε 4 bit mode στέλνουμε δύο φορές την εντολή 0x30 (function set) για 8 bit mode. Η συγκεκριμένη εντολή (μεταξύ άλλων) ρυθμίζει τον ελεγκτή να δέχεται εντολές και δεδομένα σε ένα ενιαίο κομμάτι των 8 bit. Τα 4 λιγότερο σημαντικά bit μας είναι αδιάφορα. Αν ο ελεγκτής είναι σε 8 bit mode δεν θα αλλάξει κάτι, αν όμως είναι σε 4 bit mode θα μεταβεί σε 8 bit mode. Μόλις είμαστε βέβαιοι για την μορφή που πρέπει να στέλνουμε τις εντολές μπορούμε να προχωρήσουμε με την αρχικοποίηση.

Ρουτίνα: lcd_init_sim
Αρχικοποίηση και ρυθμίσεις της οθόνης LCD όπως παρουσιάζεται παρακάτω:
DL = 0 4-bit mode
N = 1 2 lines
F = 0 5×8 dots
D = 1 display on
C = 0 cursor off
B = 0 blinking off
I/D = 1 DDRAM address auto increment
SH = 0 shift of entire display off
Είσοδος: -
Έξοδος: -
Καταχωρητές: r25:r24
Καλούμενες υπορουτίνες: *wait_msec, wait_usec, lcd_command_sim*

```
lcd_init_sim:
push r24                ; αποθήκευσε τους καταχωρητές r25:r24 γιατί τους
push r25                ; αλλάζουμε μέσα στη ρουτίνα

ldi r24, 40             ; Όταν ο ελεγκτής της lcd τροφοδοτείται με
ldi r25, 0              ; ρεύμα εκτελεί την δική του αρχικοποίηση.
rcall wait_msec         ; Αναμονή 40 msec μέχρι αυτή να ολοκληρωθεί.

ldi r24, 0x30           ; εντολή μετάβασης σε 8 bit mode
out PORTD, r24          ; επειδή δεν μπορούμε να είμαστε βέβαιοι
sbi PORTD, PD3          ; για τη διαμόρφωση εισόδου του ελεγκτή
cbi PORTD, PD3          ; της οθόνης, η εντολή αποστέλλεται δύο φορές
ldi r24, 39
ldi r25, 0              ; εάν ο ελεγκτής της οθόνης βρίσκεται σε 8-bit mode
rcall wait_usec         ; δεν θα συμβεί τίποτα, αλλά αν ο ελεγκτής έχει διαμόρφωση
                        ; εισόδου 4 bit θα μεταβεί σε διαμόρφωση 8 bit

push r24                ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25                ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24,low(1000)       ; πρόσβασης
ldi r25,high(1000)
rcall wait_usec
pop r25
pop r24                 ; τέλος τμήμα κώδικα

ldi r24, 0x30
out PORTD, r24
sbi PORTD, PD3
cbi PORTD, PD3
ldi r24,39
ldi r25,0
rcall wait_usec
```

```
push r24                ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25                ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24 ,low(1000)      ; πρόσβασης
ldi r25 ,high(1000)
rcall wait_usec
pop r25
pop r24                 ; τέλος τμήμα κώδικα

ldi r24,0x20            ; αλλαγή σε 4-bit mode
out PORTD, r24
sbi PORTD, PD3
cbi PORTD, PD3
ldi r24,39
ldi r25,0
rcall wait_usec

push r24                ; τμήμα κώδικα που προστίθεται για τη σωστή
push r25                ; λειτουργία του προγραμματος απομακρυσμένης
ldi r24 ,low(1000)      ; πρόσβασης
ldi r25 ,high(1000)
rcall wait_usec
pop r25
pop r24                 ; τέλος τμήμα κώδικα

ldi r24,0x28            ; επιλογή χαρακτήρων μεγέθους 5x8 κουκίδων
rcall lcd_command_sim   ; και εμφάνιση δύο γραμμών στην οθόνη

ldi r24,0x0c            ; ενεργοποίηση της οθόνης, απόκρυψη του κέρσορα
rcall lcd_command_sim

ldi r24,0x01            ; καθαρισμός της οθόνης
rcall lcd_command_sim

ldi r24, low(1530)
ldi r25, high(1530)
rcall wait_usec

ldi r24 ,0x06           ; ενεργοποίηση αυτόματης αύξησης κατά 1 της διεύθυνσης
rcall lcd_command_sim   ; που είναι αποθηκευμένη στον μετρητή διευθύνσεων και
                        ; απενεργοποίηση της ολίσθησης ολόκληρης της οθόνης
pop r25                 ; επανάφερε τους καταχωρητές r25:r24
pop r24
ret
```

Παράδειγμα (απεικόνιση του χαρακτήρα ‘A’ στην οθόνη):

```
main:
ldi r24, low(RAMEND)
out SPL, r24
ldi r24, high(RAMEND)
out SPH, r24            ; αρχικοποίηση stack pointer
ser r24
out DDRD, r24           ; αρχικοποίηση PORTD που συνδέεται η οθόνη, ως έξοδος
```

```
clr r24

rcall lcd_init_sim          ; αρχικοποίηση οθόνης

ldi r24, 'A'
rcall lcd_data_sim          ; αποστολή ενός byte δεδομένων στον ελεγκτή της οθόνης lcd

jmp main                    ;
```

Τα ζητούμενα της 3ης εργαστηριακής άσκησης (2η AVR)

Ζήτημα 3.1 Γράψτε ένα πρόγραμμα «ηλεκτρονικής κλειδαριάς» το οποίο να ανάβει όλα τα leds **PB0-7** για 4 sec συνολικά, μόνο όταν πατηθούν στη σειρά τα δύο πλήκτρα στο **keypad 4×4** που αντιστοιχούν στο διψήφιο αριθμό της ομάδας σας (π.χ. 09). Αν δεν έχουν δοθεί οι δύο σωστοί αριθμοί να αναβοσβήνει (χρόνος ~0.5 sec αναμμένο και ~0.5 sec σβησμένο) τα leds **PB0-7** επίσης για 4 sec. Μετά το πάτημα δύο αριθμών το πρόγραμμα να συνεχίσει να διαβάζει και άλλους αριθμούς από το keypad αλλά να τους αγνοεί. Το πρόγραμμα αυτό να είναι συνεχόμενης λειτουργίας. Δώστε το διάγραμμα ροής και το πρόγραμμα σε **C**. Επίσης η ρουτίνα **keypad_to_ascii** να δοθεί σε μορφή συνάρτησης C για να αξιοποιηθεί από το πρόγραμμα σε C.

Ζήτημα 3.2 Να ξαναγραφεί το παραπάνω πρόγραμμα του Ζητήματος 3.1 σε **assembly** αλλά να προσθέσετε την εξής λειτουργία: Να απεικονίζεται στο **LCD display** το μήνυμα **ALARM ON** στην περίπτωση που δοθεί λάθος διψήφιος αριθμός από το πληκτρολόγιο. Σε διαφορετική περίπτωση να αναγράφει **WELCOME XX,** όπου XX ο αριθμός της ομάδας.

Περιορισμοί προγράμματος απομακρυσμένης πρόσβασης:

1) Μπορείτε να χρησιμοποιήσετε τα LED της PORTB αλλά κανένα από τα Button ή μπορείτε να έχετε μονάχα 1 Button ενεργοποιημένο κάθε χρονική στιγμή και κανένα από τα LED.
2) Δεν μπορείτε για οποιαδήποτε PORT να χρησιμοποιείται και το LED και το BUTTON.
3) Δεν μπορείτε να χρησιμοποιήσετε το keypad αν δεν το διαβάζει ο ATMEGA16 με την ρουτίνα scan_keypad_rising_edge_sim αλλιώς το κουμπί που πατήσατε θα παραμείνει κόκκινο και θα πρέπει να κάνετε reset στο σύστημα.