



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2024-2025

ΑΘΗΝΑ 10/10/2025

**1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"**

Ανάπτυξη κώδικα για το μικροελεγκτή ATmega328 και προσομοίωση της εκτέλεσης του στο αναπτυξιακό περιβάλλον MPLAB X

Εξέταση – Επίδειξη: Παρασκευή 17/10/2025.

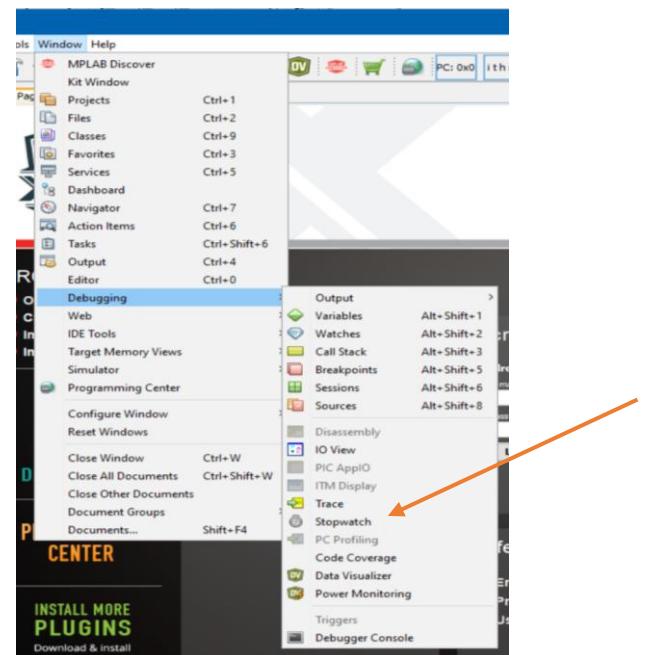
Προθεσμία για παράδοση Έκθεσης: Τρίτη 21/10/2025 (23:59)

Ζήτημα 1.1

Να υλοποιηθεί κώδικας assembly, για τον μικροελεγκτή ATmega328, στη μορφή που εμφανίζεται παρακάτω, ο οποίος να παράγει ρυθμιζόμενες χρονικές καθυστερήσεις.

```
...
rcall wait_x_msec
...
wait_x_msec:
...
ret
```

- Η εκτέλεση της εντολής rcall wait_x_msec να διαρκεί χρόνο x msec, όπου x είναι ένας αριθμός από 1 έως 65535, ο οποίος είναι αποθηκευμένος στο ζεύγος των καταχωρητών r24(Low byte) και r25(High byte).
- Το τμήμα αυτό του κώδικα να ενσωματωθεί σε ένα ολοκληρωμένο πρόγραμμα και να γίνει προσομοίωση σωστής λειτουργίας, στον αναπτυξιακό περιβάλλον Microchip MPLAB X.
- Στο τμήμα του κώδικα που γίνεται μέτρηση του χρόνου να ενσωματωθεί η εντολή “sbiw” η οποία χειρίζεται ένα ζεύγος καταχωρητών και όχι έναν απλό καταχωρητή.



Ο χρόνος εκτέλεσης της προσομοίωσης, εξαρτάται από τον υπολογιστή που εκτελεί την προσομοίωση και είναι διαφορετικός από το χρόνο εκτέλεσης του κώδικα από το φυσικό μικροελεγκτή.

Οπότε για να μετρηθούν οι χρόνοι εκτέλεσης τμημάτων του κώδικα, θα χρησιμοποιηθεί το εργαλείο Stopwatch, του MPLAB X, το οποίο ενεργοποιείται όπως φαίνεται στο σχήμα δίπλα.

Ζήτημα 1.2

Να υλοποιηθεί κώδικας assembly, για τον μικροελεγκτή ATmega328, για τον υπολογισμό των λογικών συναρτήσεων:

$$F0 = (A' \cdot B + B' \cdot D)'$$

$$F1 = (A+C) \cdot (B+D)$$

Ο υπολογισμός των συναρτήσεων να εισαχθεί σε ένα loop, το όποιο θα εκτελεστεί 6 φορές. Σε κάθε κύκλο η μεταβλητή A θα αυξάνεται κατά 0x01, η μεταβλητή B θα αυξάνεται κατά 0x02, η μεταβλητή C θα αυξάνεται κατά 0x03 και η μεταβλητή D θα αυξάνεται κατά 0x04.

Οι μεταβλητές A, B, C, D είναι μεγέθους ενός byte και έχουν αρχικές τιμές A=0x52, B=0x42, C=0x22 και D=0x02.

Να γίνει προσομοίωση στο MPLAB X και να συμπληρωθεί ο παρακάτω πίνακας:

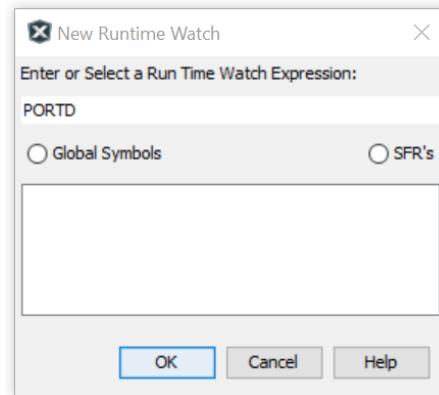
A	B	C	D	F0	F1
0x52	0x42	0x22	0x02		

Ζήτημα 1.3

Να υλοποιηθεί κώδικας assembly, για τον μικροελεγκτή ATmega328, ο οποίος να ελέγχει ένα αυτοματισμό βαγονέτου που κινείται συνεχώς, αρχικά από δεξιά προς τα αριστερά και στη συνέχεια αντίστροφα.

Το βαγονέτο να προσομοιώνεται με ένα bit της θύρας εξόδου PORTD που κινείται συνεχώς από το LSb προς το MSb και αντίστροφα.

Για να είναι εφικτή η παρακολούθηση της αλλαγής θέσης του bit της PORTD, κατά τη διάρκεια της εκτέλεσης της προσομοίωσης, είναι απαραίτητο να εισαχθεί ένα Run Time Watch, στο MPLAB X, για την PORTD, όπως φαίνεται στην εικόνα δίπλα.



- Η κίνησή του βαγονέτου κατά μία θέση, θα γίνεται κάθε 2 sec περίπου.
- Η κατεύθυνση της κίνησης να αποθηκεύεται στο T flag του SREG.
- Το βαγονέτο, κάθε φορά που αλλάζει κατεύθυνση, θα κάνει μία πρόσθετη στάση 1 sec περίπου δηλ. θα παραμένει στα άκρα 3 sec περίπου.
- Για τη δημιουργία των χρονικών καθυστερήσεων να χρησιμοποιηθεί ο κώδικας που υλοποιήθηκε στο ζήτημα 1.1
- Ο χρόνος εκτέλεσης της προσομοίωσης, εξαρτάται από τον υπολογιστή που εκτελεί την προσομοίωση. Οπότε οι διάφοροι χρόνοι θα επιτευχθούν κατόπιν δοκιμών.



**Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ**

ΑΘΗΝΑ, 17/10/2025

**2^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικρούπολογιστών"
Χρήση εξωτερικών διακοπών στον Μικροελεγκτή AVR**

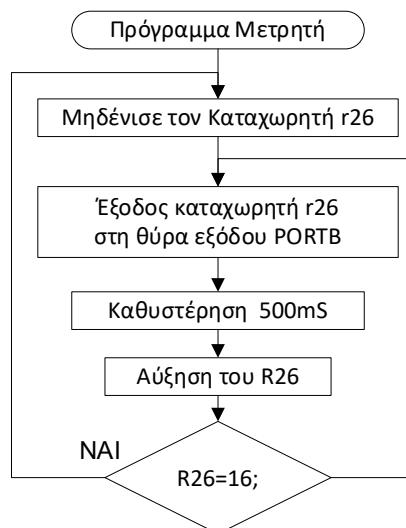
**Εξέταση – Επίδειξη: Παρασκευή 24/10/2025
Προθεσμία για παράδοση Έκθεσης: Τρίτη 28/10/2025 (23:59)**

Εισαγωγή

Μια χρήσιμη δυνατότητα των Μικροελεγκτών είναι η άμεση ανταπόκρισή τους σε εξωτερικές συνθήκες. Η ανταπόκριση αυτή επιτυγχάνεται με την εκμετάλλευση του συστήματος διακοπών τους. Κάθε μικροελεγκτής είναι εφοδιασμένος με μια ή περισσότερες εισόδους διακοπών. Η ενεργοποίηση μιας εισόδου διακοπής υποχρεώνει το μικροελεγκτή να σταματά άμεσα την τρέχουσα εργασία και να εκτελεί τον κώδικα που υπάρχει σε μια προκαθορισμένη διεύθυνση, που ονομάζεται διάνυσμα διακοπής. Στο σημείο αυτό συνδέεται συνήθως μια ρουτίνα εξυπηρέτησης διακοπής (διαφορετική για κάθε εφαρμογή). Μετά το τέλος της ρουτίνας εξυπηρέτησης διακοπής, ο Μικροελεγκτής συνεχίζει την εργασία που διέκοψε, επιστρέφοντας στο σημείο ακριβώς που είχε διακοπεί. Στην συνέχεια δίνεται ένα παράδειγμα προγράμματος μετρητή που διακόπτεται από την εξωτερική διακοπή INT0. Η διακοπή αυτή είναι συνδεδεμένη στο 3^o bit της PORTD (PD2).

Παράδειγμα 2.1 Ένα πρόγραμμα μέτρησης που διακόπτεται από την INT0

Αρχικά εισάγουμε ένα πρόγραμμα που μετράει δυαδικά από το 0 έως το 15 σε συνεχόμενη λειτουργία με καθυστέρηση 500 mS σε κάθε μέτρηση. Το αποτέλεσμα της μέτρησης εμφανίζεται στα Leds της θύρας PB3-PB0. Το διάγραμμα ροής του φαίνεται στο σχήμα 2.1a.



Σχήμα 2.1a Πρόγραμμα μετρητή.

Το πρόγραμμα φαίνεται στο σχήμα 2.1β παρακάτω:

Σχήμα 2.1β Πρόγραμμα μετρητή

```
.include "m328PBdef.inc"      ;ATmega328P microcontroller definitions

.equ FOSC_MHZ=16              ; Microcontroller operating frequency in MHz
.equ DEL_mS=500                ; Delay in mS (valid number from 1 to 4095)

.equ DEL_NU=FOSC_MHZ*DEL_mS ; delay_mS routine: (1000*DEL_NU+6) cycles

;Init Stack Pointer
ldi r24, LOW(RAMEND)
out SPL, r24
ldi r24, HIGH(RAMEND)
out SPH, r24

;Init PORTB as output
ser r26
out DDRB , r26

loop1:
clr r26
loop2:
out PORTB, r26

ldi r24, low(DEL_NU)    ;
ldi r25, high(DEL_NU)   ; Set delay (number of cycles)
rcall delay_mS          ;

inc r26

cpi r26, 16             ; compare r26 with 16
breq loop1
rjmp loop2

; delay of 1000*F1+6 cycles (almost equal to 1000*F1 cycles)
delay_mS:

; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
ldi r23, 249            ; (1 cycle)
loop_inn:
dec r23                  ; 1 cycle
nop                      ; 1 cycle
brne loop_inn            ; 1 or 2 cycles

sbiw r24 ,1              ; 2 cycles
brne delay_mS            ; 1 or 2 cycles

ret                      ; 4 cycles
```

Εξωτερικές διακοπές στον ATmega328PB

Για τη λειτουργία του συστήματος διακοπών κάθε Μικροελεγκτή είναι απαραίτητη αρχικά η ενεργοποίηση σημαιών και επιλογών, που καθορίζουν τον ακριβή τρόπο λειτουργίας.

Στον Μικροελεγκτή AVR ATmega328PB, η επιλογή του επιπέδου ενεργοποίησης των εξωτερικών διακοπών γίνεται δια μέσω του καταχωρητή EICRA (offset 0x69), γράφοντας κατάλληλες τιμές στα τέσσερα λιγότερα σημαντικά ψηφία, σύμφωνα με τους παρακάτω πίνακες:

EICRA:

				ISC11	ISC10	ISC01	ISC00
--	--	--	--	-------	-------	-------	-------

ISC11	ISC10	Περιγραφή
0	0	Διακοπή στη χαμηλή στάθμη του INT1
0	1	Διακοπή σε κάθε αλλαγή στάθμης του INT1
1	0	Διακοπή στην κατερχόμενη ακμή του INT1
1	1	Διακοπή στην ανερχόμενη ακμή του INT1

ISC01	ISC00	Περιγραφή
0	0	Διακοπή στη χαμηλή στάθμη του INT0
0	1	Διακοπή σε κάθε αλλαγή στάθμης του INT0
1	0	Διακοπή στην κατερχόμενη ακμή του INT0
1	1	Διακοπή στην ανερχόμενη ακμή του INT0

Η επίτρεψη των εξωτερικών διακοπών ενεργοποιείται γράφοντας στον καταχωρητή EIMSK (offset 0x3D) την τιμή 1 στο ψηφίο που αντιστοιχεί στην είσοδο διακοπής που επιθυμούμε να επιτρέψουμε, σύμφωνα με το παρακάτω σχήμα:

EIMSK:

						INT1	INT0
--	--	--	--	--	--	------	------

Ο καταχωρητής EIFR(offset 0x3C) περιέχει τις σημαίες των διακοπών INT0 και INT1. Κάθε μια από αυτές τίθεται όταν συμβεί η αντίστοιχη διακοπή και μηδενίζεται μετά την εξυπηρέτηση της διακοπής.

EIFR:

						INTF1	INTF0
--	--	--	--	--	--	-------	-------

Επίσης, απαραίτητη είναι και η εκτέλεση της εντολής sei, που επιτρέπει γενικώς την πρόκληση διακοπών.

Στον Μικροελεγκτή AVR ATmega328P, ο ακροδέκτης **PD2**, εκτός από ακροδέκτης εισόδου/εξόδου γενικής χρήσης, λειτουργεί εναλλακτικά και ως εξωτερική είσοδος της διακοπής INT0. Ομοίως ο ακροδέκτης **PD3** λειτουργεί εναλλακτικά και ως εξωτερική είσοδος της διακοπής INT1.

Η ροутίνα εξυπηρέτησης της διακοπής πρέπει να δηλωθεί στην κατάλληλη διεύθυνση του προγράμματος. Το διάνυσμα της διακοπής INT0 είναι 0x002 και το διάνυσμα της διακοπής INT1 είναι 0x004.

Για παράδειγμα, το παρακάτω τμήμα κώδικα (Σχήμα 2.2) ενεργοποιεί διακοπές στην είσοδο INT0 κατά την ανερχόμενη ακμή. Πρέπει να δοθεί προσοχή στο γεγονός ότι ο I/O καταχωρητής EICRA έχει η διεύθυνση μεγαλύτερη από 0x60, οπότε χρησιμοποιείται η εντολή STS και όχι η εντολή OUT.

Στη ρουτίνα εξυπηρέτησης της εξωτερική διακοπής INT0, ανάβουν τα led της θύρας PORTB για 500mS και στην συνέχεια συνεχίζεται η μέτρηση από το σημείο που είχε μείνει.

Η ενεργοποίηση της διακοπής INT0 δίνεται στο Σχήμα 2.2 και η ρουτίνα εξυπηρέτησης διακοπής στο Σχήμα 2.3.

Σχήμα 2.2 Εντολές που ενεργοποιούν τη διακοπή INT0

```
.org 0x0
rjmp reset
.org 0x2
rjmp ISR0

reset:
; Interrupt on rising edge of INT0 pin
ldi r24,(1 << ISC01) | ( 1 << ISC00)
sts EICRA, r24
;  

;Enable the INT0 interrupt(PD2)
ldi r24, (1 << INT0)
out EIMSK, r24

sei           ;Sets the Global Interrupt Flag
```

Σχήμα 2.3 Ρουτίνα εξυπηρέτησης διακοπής

```
ISR0:
push r25          ;
push r24          ;
in r24, SREG      ; Save r24, r25, SREG
push r24          ;

ser r24
out PORTB , r24

ldi r24, low(16*500)    ;
ldi r25, high(16*500)   ; Set delay (number of cycles)
rcall delay_ms

pop r24          ;
out SREG , r24      ; Restore r24, r25, SREG
pop r24          ;
pop r25          ;

reti
```

Το Φαινόμενο της Αναπήδησης (ή Σπινθηρισμού)

Εάν ο χρόνος που διαρκεί η εκτέλεση μιας ρουτίνας εξυπηρέτησης διακοπής είναι πολύ μικρός, ενδέχεται να εμφανιστεί το φαινόμενο της αναπήδησης, λόγω σπινθηρισμού στον πιεστικό διακόπτη που προκαλεί την εξωτερική διακοπή. Συγκεκριμένα, ένας πιεστικός διακόπτης μπορεί να δώσει περισσότερα του ενός σήματα διακοπής τόσο κατά την πίεση του όσο και κατά την απελευθέρωση του, οπότε μετά την ολοκλήρωση της ρουτίνας εξυπηρέτησης της διακοπής, μπορεί να καταφθάσουν και άλλα σήματα διακοπής που οφείλονται στο ίδιο πάτημα ή απελευθέρωση του πιεστικού διακόπτη. Μια λύση για το πρόβλημα αυτό για τον μικροελεγκτή ATmega328PB είναι ο έλεγχος του καταχωρητή EIFR μέσα στη ρουτίνα εξυπηρέτησης διακοπής.

Τα ψηφία INTF1 και INTF0 του καταχωρητή EIFR τίθενται αυτόματα από το μικροελεγκτή όταν υπάρχει αίτηση εξωτερικής διακοπής INT1 και INT0 αντίστοιχα, και μηδενίζονται όταν εξυπηρετηθεί η αίτηση. Τα ψηφία INTF1 και INTF0 μπορούν να μηδενιστούν και από τον κώδικα, με εγγραφή λογικού 1 στο αντίστοιχο ψηφίο.

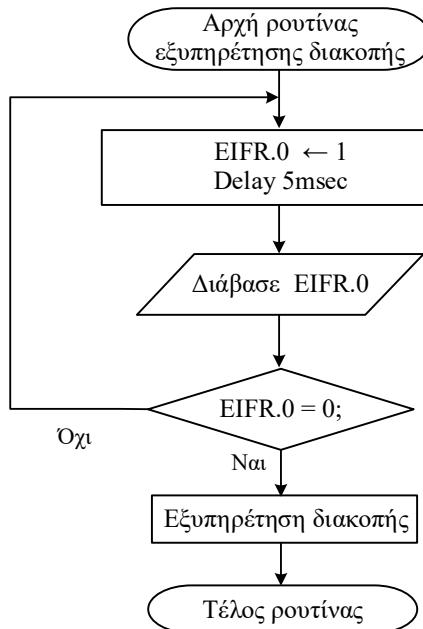
Π.χ. το INT0 μπορεί να μηδενιστεί με τις παρακάτω εντολές:

```
ldi r24, (1 << INTF0)  
out EIFR, r24
```

Ένας απλός αλγόριθμος αποφυγής της αναπήδησης είναι ο εξής:

Στην αρχή της ρουτίνας εξυπηρέτησης διακοπής μηδενίζεται το ψηφίο INTF1 ή INTF0. Μετά από ένα μικρό χρονικό διάστημα (π.χ. 5 msec) το ψηφίο ελέγχεται ξανά και αν είναι λογικό 1 τότε κάποιος σπινθηρισμός έχει δώσει νέο σήμα διακοπής για το ίδιο πάτημα και το ψηφίο μηδενίζεται ξανά. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου το ψηφίο σταθεροποιηθεί σε λογικό 0, οπότε έχουν σταματήσει οι αναπηδήσεις, και ολοκληρώνεται η ρουτίνα εξυπηρέτησης.

Τα παραπάνω, για την περίπτωση της εξωτερικής εισόδου INT0 φαίνονται και στο επόμενο λογικό διάγραμμα.



Σχήμα 2.4 Λογικό διάγραμμα αποφυγής σπινθηρισμού (debouncing) πλήκτρου διακοπής INT0.

Παράδειγμα 2.2 Ένα πρόγραμμα με εξωτερικά Interrupts σε γλώσσα C

Το πρόγραμμα που φαίνεται στο παρακάτω σχήμα (Σχήμα 2.5) αποτελείται από τον κώδικα της κυρίας ρουτίνας, ο οποίος αναβοσβήνει συνεχώς τα led της PORTB, με καθυστέρηση 500mS. Επίσης ενεργοποιεί τις εξωτερικές διακοπές INT0 και INT1 κατά την ανερχόμενη ακμή τάσης στα αντίστοιχα pins. Οι ρουτίνες εξυπηρέτησης των διακοπών ανάβουν τα led της PORTC για 2 Sec.

Σχήμα 2.5 Κώδικας που αναβοσβήνει τα led της PORTB και διακόπτεται από τα INT0 και INT1.

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>

ISR (INT0_vect)           //External INT0 ISR
{
    PORTC = 0xFF;
    _delay_ms(2000);
    PORTC = 0x00;
}

ISR (INT1_vect)           //External INT1 ISR
{
    PORTC = 0xFF;
    _delay_ms(2000);
    PORTC = 0x00;
}

int main()
{
    //Interrupt on rising edge of INT0 and INT1 pin
    EICRA=(1 << ISC01) | ( 1 << ISC00) | (1 << ISC11) | ( 1 << ISC10);
    //Enable the INT0 interrupt(PD2), INT1 interrupt(PD3)
    EIMSK=(1 << INT0)|(1 << INT1);
    sei();          // Enable global interrupts

    DDRB=0xFF;      //Set PORTB as output
    DDRC=0xFF;      //Set PORTC as output
    PORTC=0x00;

    while(1)
    {
        PORTB = 0x00;
        _delay_ms(500);
        PORTB = 0xFF;
        _delay_ms(500);
    }
}
```

Τα ζητούμενα της 2^{ης} εργαστηριακής άσκησης

Ζήτημα 2.1

A) Στον κώδικα του μετρητή του σχήματος 2.1, να προστεθεί ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής **INT1** (PD3), σε γλώσσα Assembly, η οποία να απαριθμεί το πλήθος των διακοπών INT1 από 0 έως 31. Όταν φτάσει 31 να αρχίζει ξανά από το μηδέν. Όσο είναι πατημένο το μπουτόν **PD1** (λογικό 0) η μέτρηση των διακοπών παγώνει. Όταν το μπουτόν **PD1** αφεθεί ξανά, η μέτρηση συνεχίζεται από το σημείο που είχε μείνει. Το πλήθος των εξωτερικών διακοπών να απεικονίζεται, σε δυαδική μορφή, στα leds **PC5-PC1**.

B) Επειδή υπάρχει η πιθανότητα να εμφανιστεί το φαινόμενο του σπινθηρισμού, με το πάτημα του μπουτόν **PD3** του προηγούμενου παραδείγματος, να προστεθούν οι κατάλληλες εντολές στη ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής **INT1**, ώστε να αποφευχθεί το φαινόμενο αυτό.

Ζήτημα 2.2

A) Να υλοποιηθεί κώδικας Assembly για το μικροελεγκτή ATmega328PB, αντίστοιχο με τον κώδικα του μετρητή του σχήματος 2.1, με τη διαφορά ότι η μέτρηση θα είναι 0 έως 31, η απεικόνιση θα γίνεται στα leds **PC5-PC1** και ο χρόνος καθυστέρησης μεταξύ των εναλλαγών θα κυμαίνεται στα 1000 mS.

B) Η ρουτίνα εξυπηρέτησης της εξωτερικής διακοπής **INT0** (PD2) να τροποποιηθεί έτσι ώστε όταν ενεργοποιείται να ανάβει τόσα LEDs της θύρας **PORTC** αρχίζοντας από το **LSB** όσο το πλήθος των 4 μπουτόν **PB4 – PB1** που είναι πατημένα.

Ζήτημα 2.3

Να υλοποιηθεί αυτοματισμός που να ελέγχει το άναμμα και το σβήσιμο ενός φωτιστικού σώματος. Όταν πατάμε το push button **PD3** (δηλαδή με την ενεργοποίηση της **INT1**) να ανάβει το led **PB3** της θύρας **PORTB** (που υποθέτουμε ότι αντιπροσωπεύει το φωτιστικό σώμα). Το led θα σβήνει μετά από 4 sec, εκτός και αν ενδιάμεσα υπάρξει νέο πάτημα του **PD3**, οπότε και ο χρόνος των 4 sec θα ανανεώνεται. Κάθε φορά που γίνεται ανανέωση να ανάβουν τα led της θύρας **PORTB** (PB5-PB0) για 1 sec, μετά να σβήνουν εκτός από το led **PB3** που παραμένει συνολικά για 4 sec εκτός και αν ανανεωθεί. Ο κώδικας να δοθεί σε **Assembly** και σε **C**.



ΑΘΗΝΑ, 24/10/2025

**3^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικρούπολογιστών"
Χρήση των Timers και του ADC στον AVR**

**Εξέταση – Επίδειξη: Παρασκευή 31/10/2025
Προθεσμία για παράδοση Έκθεσης: Τρίτη 4/11/2025 (23:59)**

Χρονιστές(Timers)

Ένα πλεονέκτημα των σύγχρονων μικροελεγκτών είναι η ενσωμάτωση στην ίδια ψηφίδα χρήσιμων περιφερειακών συσκευών, όπως οι χρονιστές, με σύνδεση στο σύστημα διακοπών του Μικροελεγκτή. Οι χρονιστές είναι καταχωρητές που μεταβάλλονται με την πάροδο του χρόνου και μπορούν να προγραμματιστούν να προκαλέσουν διακοπή στον μικροελεγκτή σε συγκεκριμένα χρονικά διαστήματα.

Ο μικροελεγκτής ATmega328PB διαθέτει 8-ψήφιους και 16-ψήφιους χρονιστές. Στους χρονιστές αυτούς μπορεί να τοποθετηθεί μια αρχική τιμή η οποία αυξάνεται με επιλεγμένη συχνότητα. Όταν ένας χρονιστής υπερχειλίσει, μπορεί να δημιουργήσει κατάλληλο σήμα διακοπής.

Timer/Counter1(TCNT1)

Ο Timer/Counter1(TCNT1) είναι 16 bit. Το TCNT1L με διεύθυνση 0x84 είναι το Low Byte και το TCNT1H με διεύθυνση 0x85 είναι το Hi Byte.

Επιλογή συχνότητας λειτουργίας του Timer/Counter1(TCNT1)

Για την επιλογή της συχνότητας λειτουργίας του Timer/Counter1(TCNT1) χρησιμοποιούνται τα flags CS10, CS11 και CS12 του καταχωρητή TCCR1B σύμφωνα με τον Πίνακα 3.1 παρακάτω:

TCCR1B (Διεύθυνση: 0x81)

Bit	7	6	5	4	3	2	1	0
Access	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

CS12	CS11	CS10	Περιγραφή σήματος εισόδου χρονιστή
0	0	0	Κανένα. Χρονιστής σταματημένος
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	Κατερχόμενη ακμή εξωτερικού σήματος ακροδέκτη T1
1	1	1	Ανερχόμενη ακμή εξωτερικού σήματος ακροδέκτη T1

Πίνακας 3.1: Επιλογή συχνότητας λειτουργίας του Timer/Counter1(TCNT1)

Για παράδειγμα, με τις παρακάτω εντολές επιλέγεται συχνότητα λειτουργίας του Timer/Counter1(TCNT1) ίση με τη 1/1024 της συχνότητας ρολογιού του μικροελεγκτή (δηλαδή, κάθε 1024 κύκλους ρολογιού αυξάνεται κατά 1 ο TCNT1).

```
ldi r24, (I<<CS12) | (0<<CS11) | (1<<CS10) ; CK/1024
sts TCCR1B, r24
```

Στην εκπαιδευτική κάρτα ntuAboard_G1 με συχνότητα ρολογιού 16MHz, η επιλογή αυτή ισοδυναμεί με συχνότητα αύξησης του TCNT1 ίση με 16MHz/1024=15625Hz.

Αν με αυτές τις επιλογές θέλουμε ο TCNT1 να δημιουργήσει σήμα διακοπής υπερχείλισης μετά από 3sec, πρέπει να ρυθμιστεί για μέτρημα $3 \times 15625 = 46875$ κύκλων. Επειδή η υπερχείλιση γίνεται μετά από 65536 κύκλους (16 ψηφία), θα πρέπει η αρχική τιμή που θα του δοθεί πριν αρχίσει τη μέτρηση προς τα πάνω να είναι $65535 - 46875 = 18660$. Αυτό γίνεται με τον παρακάτω κώδικα:

```
ldi r24, HIGH(18660) ; αρχικοποίηση του TCNT1
out TCNT1H, r24 ; για υπερχείλιση μετά από 3 sec
ldi r24, LOW(18660)
out TCNT1L, r24
```

Ανάγνωση/Εγγραφή του Timer/Counter1(TCNT1)

Για μέγιστη ασφάλεια, τα δύο τμήματα του μετρητή TCNT1, TCNT1H και TCNT1L πρέπει να διαβάζονται αδιαίρετα, χωρίς να μεσολαβήσει για παράδειγμα κάποια άλλη διακοπή. Για το λόγο αυτό ο μικροελεγκτής κάνει μια ειδική διαδικασία. Όταν μια τιμή γράφεται στον TCNT1H, αυτή τοποθετείται στον προσωρινό καταχωρητή TEMP. Στη συνέχεια, όταν γραφεί τιμή στον TCNT1L η τιμή που υπάρχει στον TEMP συνδυάζεται με αυτή και τα 16 ψηφία γράφονται ταυτόχρονα σε όλο το μήκος του TCNT1. Κατά την ανάγνωση, όταν διαβάζεται μια τιμή από τον TCNT1L αυτή τοποθετείται στον επιλεγμένο καταχωρητή του μικροελεγκτή και ταυτόχρονα η τιμή του TCNT1H μεταφέρεται στον καταχωρητή TEMP. Όταν στην συνέχεια διαβαστεί και ο TCNT1H, μεταφέρεται στον επιλεγμένο καταχωρητή η τιμή που έχει τοποθετηθεί στον TEMP.

Με βάση αυτή τη διαδικασία κατά την εγγραφή πρέπει πάντα να γράφεται πρώτα η τιμή στον TCNT1H και μετά στον TCNT1L ενώ κατά την ανάγνωση πρέπει πάντα να διαβάζεται πρώτα ο TCNT1L και μετά ο TCNT1H. Σε πολύπλοκες εφαρμογές πριν την πρόσβαση στον TCNT1 ενδέχεται να είναι απαραίτητη η απενεργοποίηση των διακοπών.

Παρόμοια τεχνική χειρισμού μπορεί να χρησιμοποιηθεί και για άλλους 16-bit καταχωρητές όπως οι OCR1 και ICR1.

Διακοπές υπερχείλισης του Timer/Counter1(TCNT1)

Η επίτρεψη της διακοπής υπερχείλισης του Timer/Counter1(TCNT1)1 ρυθμίζεται δια μέσω του flag TOIE1 του καταχωρητή TIMSK1 σύμφωνα με το παρακάτω σχήμα:

TIMSK1 (Διεύθυνση: 0x6F)

Bit	7	6	5	4	3	2	1	0
Access			ICIE1			OCIE1B	OCIE1A	TOIE1
			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Γράφοντας 1 στο ψηφίο TOIE1 επιτρέπονται διακοπές υπερχείλισης του Timer/Counter1(TCNT1), εφόσον επιτραπούν γενικά οι διακοπές με την εντολή sei.

Για παράδειγμα, με τις παρακάτω εντολές επιτρέπεται η διακοπή υπερχείλισης του μετρητή TCNT1.

```
ldi r24, (1<<TOIE1) ; ενεργοποίηση διακοπής υπερχείλισης του TCNT1
sts TIMSK1, r24 ;
```

Η θέση μνήμης του διανύσματος διακοπής του Timer1 φαίνεται στον παρακάτω κώδικα.

.org 0x1A

rjmp ISR_TIMER1_OVF ; ρουτίνα εξυπηρέτησης της διακοπής υπερχείλισης του TCNT1

To flag υπερχείλισης του Timer/Counter1 (TOV1) βρίσκεται στον καταχωρητή TIFR1 όπως φαίνεται στο παρακάτω σχήμα:

TIFR1 (Διεύθυνση: 0x36)

Bit	7	6	5	4	3	2	1	0
			ICF1			OCF1B	OCF1A	TOV1
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Ρύθμιση τρόπου λειτουργίας του Timer/Counter1

Τα flags WGM10, WGM11 του καταχωρητή TCCR1A και τα flags WGM12, WGM13 του καταχωρητή TCCR1B, ο οποίος παρουσιάστηκε στη σελίδα 1 παραπάνω, χρησιμεύουν για να καθοριστεί ο τρόπος λειτουργίας του Timer/Counter1(TCNT1), σύμφωνα με τον παρακάτω πίνακα:

TCCR1A (Διεύθυνση: 0x80)

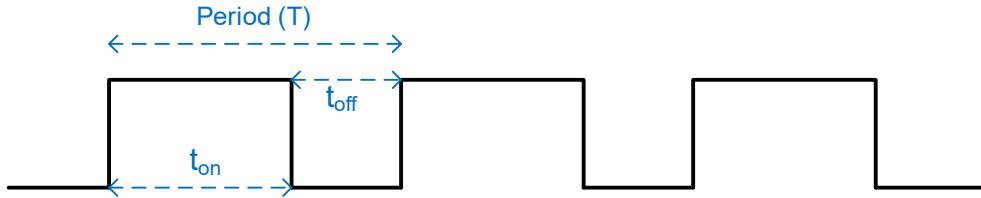
Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0			WGM11	WGM10
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Mode	WGM13	WGM12	WGM11	WGM10	Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Πίνακας 3.2: Ρύθμιση τρόπου λειτουργίας του Timer/Counter1

Διαμόρφωση εύρους παλμών (Pulse Width Modulation, PWM)

Μία PWM κυματομορφή είναι μία τετραγωνική περιοδική κυματομορφή η οποία έχει δύο τμήματα. Το τμήμα ON στο οποίο η κυματομορφή μία μέγιστη τιμή και το τμήμα OFF στο οποίο έχει μία ελάχιστη τιμή. Η περίοδος της κυματομορφής είναι σταθερή ενώ οι χρόνοι t_{on} και t_{off} μεταβάλλονται.



Σχήμα 3.1: PWM κυματομορφή

Ο βαθμός χρησιμοποίησης (Duty Cycle) συμβολίζεται με DC και ορίζεται σύμφωνα με τον τύπο:

$$DC = \frac{t_{on}}{T}$$

Διαμόρφωση εύρους παλμών με τον ATmega328PB

Ο ATmega328PB διαθέτει διάφορους μετρητές (Timer/Counters) οι οποίοι μπορούν να χρησιμοποιηθούν για την παραγωγή PWM κυματομορφών τάσης. Από αυτούς ο Timer/Counter1(TCNT1) έχει δύο εξόδους PWM και μπορεί να ρυθμιστεί ως 8-bit ή ως 16-bit. Η μία PWM έξοδος συμβολίζεται ως OC1A και μπορεί να συνδεθεί στον ακροδέκτη PB1 ενώ η άλλη PWM έξοδος συμβολίζεται ως OC1B και μπορεί να συνδεθεί στον ακροδέκτη PB2, Οι υπόλοιποι Timers/Counters του ATmega328PB έχουν παρόμοιο τρόπο λειτουργίας.

Εδώ θα εξεταστεί ένας τρόπος λειτουργίας με τον οποίο παράγεται μια PWM κυματομορφή τάσης υψηλής συχνότητας. Στη λειτουργία αυτή (Fast PWM Mode) ο Timer/Counter1(TCNT1) ανξάνεται ξεκινώντας από την τιμή BOTTOM και όταν φτάσει την τιμή TOP τότε παίρνει ξανά την τιμή BOTTOM και η διαδικασία επαναλαμβάνεται.

Η κυματομορφή εξόδου θα είναι παλμοί με σταθερή **συχνότητα (f_{PWM})** η τιμή της οποίας εξαρτάται από τη συχνότητα του ρολογιού του συστήματος (f_{clk}) και την αρχικοποίηση του prescaler όπως προκύπτει από τον παρακάτω τύπο:

$$f_{PWM} = \frac{f_{clk}}{N \cdot (1 + TOP)}$$

Η μεταβλητή N αντιπροσωπεύει την τιμή του prescaler (1, 8, 64, 256 ή 1024).

Το Duty Cycle ρυθμίζεται, δια μέσω της τιμής του καταχωρητή OCR1A. Η χαμηλότερη τιμή για τον καταχωρητή OCR1A είναι η BOTTOM, όπου η έξοδος θα είναι ένας εξαιρετικά μικρός παλμός στην αρχή κάθε περιόδου και η υψηλότερη τιμή είναι η TOP όπου η έξοδος θα είναι σταθερά υψηλή. Το Low byte του OCR1AL έχει διεύθυνση 0x88 και το Hi byte του OCR1AH έχει διεύθυνση 0x89.

Στην ανάστροφη PWM λειτουργία, οι κυματομορφές εξόδου είναι αντεστραμμένες.

Τα flags COM1A0 και COM1A1 του καταχωρητή TCCR1A, χρησιμεύουν για να καθοριστεί εάν η PWM έξοδος OC1A του Timer1 θα συνδεθεί ή όχι στον ακροδέκτη PB1, καθώς επίσης και η μέθοδος παραγωγής της PWM κυματομορφής σε αυτή την έξοδο. Για παραγωγή της μη ανάστροφης PWM κυματομορφής που περιεγράφηκε παραπάνω οι τιμές των (COM1A0, COM1A1) είναι (0, 1) ενώ για ανάστροφη λειτουργεία οι τιμές είναι (1, 1) Αντίστοιχα τα flags COM1B0 και COM1B1 χρησιμεύουν για να καθοριστεί εάν η PWM έξοδος OC1B του Timer1 θα συνδεθεί ή όχι στον ακροδέκτη PB2, καθώς επίσης και η μέθοδος παραγωγής της PWM κυματομορφής σε αυτή την έξοδο.

Για όλους τους υπολογισμούς η τιμή BOTTOM θα είναι το μηδέν και η τιμή TOP θα δίνεται από τον Πίνακα 3.2 που παρουσιάστηκε παραπάνω.

Παράδειγμα 3.1

Στη συνέχεια παρουσιάζεται ένα παράδειγμα παραγωγής παλμών PWM στον ακροδέκτη PB1, σε γλώσσα προγραμματισμού C, για την εκπαιδευτική κάρτα ntuAboard_G1. Ο κώδικας αρχικοποιεί κατάλληλα τον χρονιστή TMR1A για να λειτουργεί σε Fast PWM Mode, με μη ανάστροφη λειτουργία(non-inverting mode) και με τιμή 8 για τον prescaler. Ο TMR1B δεν χρησιμοποιείται. Στον ακροδέκτη PB1 είναι συνδεδεμένο ένα LED. Η ρουτίνα main() αυξομειώνει συνεχώς το Duty Cycle της PWM κυματομορφής αυξομειώνοντας αντίστοιχα την φωτεινότητα του LED που είναι συνδεμένο στον ακροδέκτη PB1.

```
#define F_CPU 16000000UL
#include "avr/io.h"
#include <util/delay.h>

int main ()
{
    unsigned char duty;

    //set TMR1A in fast PWM 8 bit mode with non-inverted output
    //prescale=8
    TCCR1A = (1<<WGM10) | (1<<COM1A1);
    TCCR1B = (1<<WGM12) | (1<<CS11);

    DDRB |= 0b00111111;      //set PB5-PB0 pins as output

    while (1)
    {
        for(duty=0; duty<255; duty++)
        {
            OCR1AL=duty; //increase the LED2 light intensity
            _delay_ms(10);
        }
        for(duty=255; duty>1; duty--)
        {
            OCR1AL=duty; //increase the LED2 light intensity
            _delay_ms(10);
        }
    }
}
```

Μετατροπέας αναλογικής σε ψηφιακή μορφή ADC.

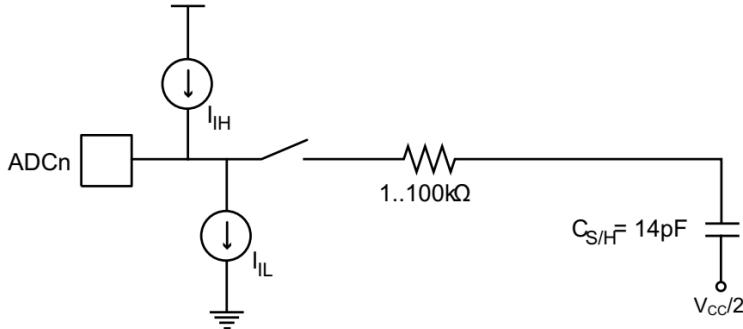
Βασικό περιφερειακό των μικροελεγκτών είναι ο μετατροπέας από Αναλογική σε Ψηφιακή μορφή (ADC). Αν V_{in} είναι η αναλογική τάση στην εισόδου του ADC με ανάλυση n-bit, V_{REF} μία τάση αναφοράς βάση της οποίας γίνεται η μετατροπή και $V_{in} \leq V_{REF}$, τότε η ψηφιακή έξοδος του ADC μετατροπέα είναι:

$$ADC = \frac{V_{in} \cdot 2^n}{V_{REF}}$$

Στην παραπάνω σχέση όλες η ποσότητες είναι ακέραιοι θετικοί αριθμοί.

Αναλογικό κύκλωμα εισόδου

Το κύκλωμα αναλογικής εισόδου του ADC απεικονίζεται στο παρακάτω σχήμα:



Σχήμα 3.2: Κύκλωμα αναλογικής εισόδου στον ADC

Μια αναλογική πηγή που εφαρμόζεται στον ακροδέκτη ADCn επηρεάζεται από τη χωρητικότητα του ακροδέκτη αυτού και από το ρεύμα διαρροής του. Όταν επιλεγεί το κανάλι αυτό για μετατροπή τότε πρέπει το σήμα του να φορτίσει τον πυκνωτή S/H δια μέσω της αντίστασης σειράς (Ολική αντίσταση στη διαδρομή εισόδου).

Το ADC είναι βελτιστοποιημένο για αναλογικά σήματα εισόδου με σύνθετη αντίσταση μέχρι 10 KΩ, όπου ο χρόνος δειγματοληψίας είναι αμελητέος. Εάν χρησιμοποιηθεί πηγή με υψηλότερη τιμή αντίστασης τότε ο χρόνος δειγματοληψίας αυξάνεται και εξαρτάται από το χρόνο που απαιτείται για να φορτιστεί ο πυκνωτής S/H.

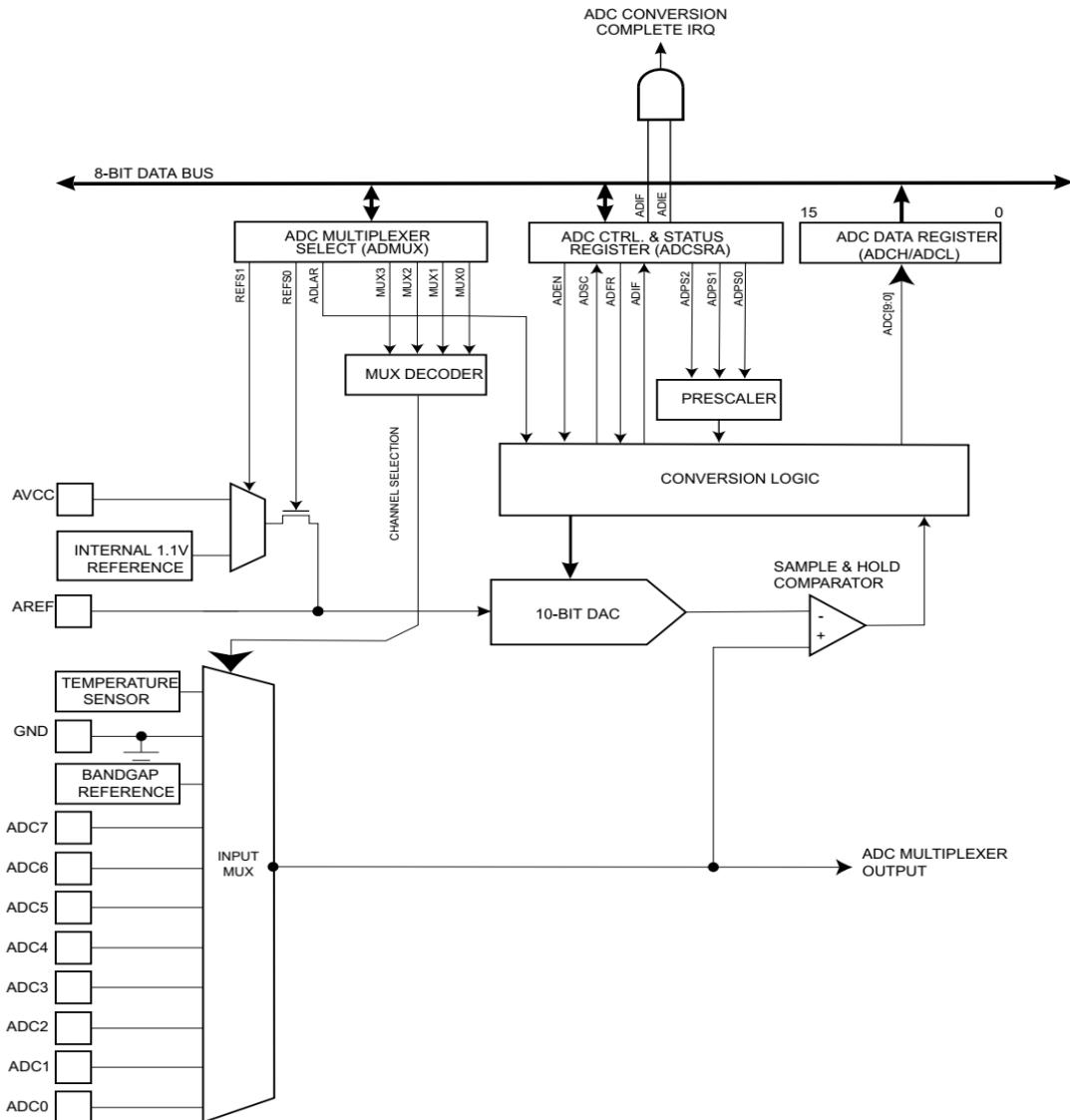
Σε κανονική λειτουργία ο χρόνος δειγματοληψίας(sampling time) είναι 1,5 κύκλος ρολογιού του ADC.

AVR ATMega328PB ADC

Ο μικροελεγκτής AVR ATMega328PB διαθέτει έναν ADC, ανάλυσης n=10 bit, που βασίζεται στη μέθοδο των διαδοχικών προσεγγίσεων.

Το κύκλωμα διαδοχικής προσέγγισης του ADC απαιτεί συχνότητα ρολογιού μεταξύ 50 kHz και 200 kHz για μέγιστη ανάλυση. Εάν απαιτείται ανάλυση χαμηλότερη από 10 bit, η συχνότητα ρολογιού εισόδου του ADC μπορεί να είναι υψηλότερη από 200 KHz.

Στο παρακάτω σχήμα φαίνεται το μπλοκ διάγραμμά του ADC μετατροπέα:



Σχήμα 3.3: Μπλοκ διάγραμμά του ADC

Αναλογικός πολυπλέκτης του ADC

Ο ADC είναι συνδεδεμένος με ένα αναλογικό πολυπλέκτη 16 εισόδων. Το κανάλι αναλογικής εισόδου που είναι κάθε φορά συνδεδεμένο στο κύκλωμα μετατροπής του ADC επιλέγεται από τα bit MUX[3:0] του καταχωρητή ADMUX, όπως φαίνεται παρακάτω:

ADMUX (Διεύθυνση: 0x7C). Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
	REFS _n	REFS _n	ADLAR		MUX _n	MUX _n	MUX _n	MUX _n
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Σχήμα 3.3: Καταχωρητής ADMUX

MUX[3:0]	ΚΑΝΑΛΙ ΕΙΣΟΔΟΥ
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Temperature sensor
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	1.1V (VBG)
1111	0V (GND)

Πίνακας 3.3: Επιλογή αναλογικών εισόδων του ADC

Στον παρακάτω πίνακα εμφανίζονται οι ακροδέκτες του μικροελεγκτή AVR ATMega328PB που μπορούν να δεχτούν αναλογικά σήματα και τα κανάλια του ADC που αντιστοιχούν. Όσοι από αυτούς τους ακροδέκτες θα χρησιμοποιηθούν από τον ADC πρέπει να οριστούν ως είσοδοι, κάνοντας χρήση του αντίστοιχου καταχωρητή DDRx.

ΑΚΡΟΔΕΚΤΗΣ	ΚΑΝΑΛΙ ΤΟΥ ADC
PC0	ADC0
PC1	ADC1
PC2	ADC2
PC3	ADC3
PC4	ADC4
PC5	ADC5
PE2	ADC6
PE3	ADC7

Πίνακας 3.4: Αναλογικές είσοδοι του ADC

Τα bit REFS[1:0] χρησιμοποιούνται για την επιλογή του V_{REF} σύμφωνα με τον επόμενο πίνακα:

REFS[1:0]	Voltage reference selection
00	AREF, internal V_{REF} turned OFF
01	AVCC with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V voltage reference with external capacitor at AREF pin

Πίνακας 3.5: Επιλογή του ADC V_{REF}

Έλεγχος της λειτουργίας του ADC

Ο έλεγχος της λειτουργίας του ADC γίνετε δια μέσω των καταχωρητών ADCSRA και ADCSRB. Ο καταχωρητής ADCSRA φαίνεται στο παρακάτω σχήμα:

ADCSRA (Διεύθυνση: 0x7A). ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPSn	ADPSn	ADPSn
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Σχήμα 3.4: Καταχωρητής ADCSRA

Στη συνέχεια παρατίθενται οι λειτουργίες αυτού του καταχωρητή:

ADEN: ADC Enable

Όταν η σημαία αυτή τεθεί τότε ο ADC ενεργοποιείται.

ADSC: ADC Start Conversion

Στη λειτουργία μεμονωμένης μετατροπής (Single Conversion mode), η σημαία αυτή πρέπει να τεθεί για να ξεκινήσει μία μετατροπή. Στη λειτουργία Free Running, η σημαία αυτή πρέπει να τεθεί για να ξεκινήσει η πρώτη μετατροπή. Η πρώτη μετατροπή διαρκεί 25 κύκλους ρολογιού του ADC ενώ οι επόμενες μετατροπές διαρκούν 13 κύκλους ρολογιού. Η πρώτη μετατροπή εκτελεί την αρχικοποίηση του ADC.

ADATE: ADC Auto Trigger Enable

Όταν η σημαία αυτή τεθεί, ενεργοποιείται η λειτουργία Auto Triggering του ADC. Το ADC θα ξεκινήσει αυτόματα μια μετατροπή σε μια θετική ακμή του επιλεγμένου σήματος σκανδαλισμού. Η πηγή σκανδαλισμού επιλέγεται ρυθμίζοντας τα bit ADTS[2:0] στον καταχωρητή ADCSRB.

ADIF: ADC Interrupt Flag

Η σημαία αυτή τίθεται αυτόματα όταν ολοκληρωθεί μια μετατροπή και ενημερωθούν οι καταχωρητές δεδομένων. Αν η τιμή της σημαίας ADIE καθώς και της σημαίας I στον SREG είναι 1 τότε θα προκληθεί διακοπή ολοκλήρωσης μετατροπής του ADC. Το ADIF μηδενίζεται από το υλικό κατά την εκτέλεση των αντίστοιχου διανύσματος χειρισμού διακοπών. Εναλλακτικά, το ADIF μηδενίζεται από τον κώδικα γράφοντας σε αυτό λογικό 1.

ADIE: ADC Interrupt Enable

Αν η τιμή της σημαίας ADIE καθώς και της σημαίας I στον SREG είναι 1 τότε θα προκληθεί διακοπή ολοκλήρωσης μετατροπής του ADC.

ADPS[2:0]: ADC Prescaler Select bits

Τα bit αυτά θέτουν ένα συντελεστή διαίρεσης μεταξύ του κεντρικού ρολογιού και της συχνότητας λειτουργίας του ADC όπως φαίνεται στον επόμενο πίνακα:

ADPS[2:0]	Division Factor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Πίνακας 3.6: ADC Prescaler Select bits

Το κύκλωμα διαδοχικής προσέγγισης του ADC, για μέγιστη ανάλυση 10 Bits, απαιτεί συγχότητα ρολογιού εισόδου μεταξύ 50 KHz και 200 KHz.

Ο καταχωρητής ADCSRB φαίνεται στο παρακάτω σχήμα:

ADCSRB (Διεύθυνση: 0x7B). ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0
		ACME				ADTSn	ADTSn	ADTSn
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

Σχήμα 3.5: Καταχωρητής ADCSRB

Στη συνέχεια παρατίθενται οι λειτουργίες αυτού του καταχωρητή:

ACME: Analog Comparator Multiplexer Enable

Χρησιμοποιείται για την επιλογή της αναλογικής εισόδου στον analog comparator, ο οποίος είναι μια περιφερειακή συσκευή, ενσωματωμένη στον μικροελεγκτή ATmega328PB.

ADTSn [2:0]: ADC Auto Trigger Source

Όταν η σημαία ADATE του καταχωρητή ADCSRA είναι 1 τότε ενεργοποιείται η λειτουργία Auto Triggering του ADC. Το ADC θα ξεκινάει αυτόματα μια μετατροπή στη θετική ακμή του σήματος σκανδαλισμού το οποίο επιλέγεται σύμφωνα με τον παρακάτω πίνακα:

ADTS[2:0]	Trigger Source
000	Free Running mode
001	Analog Comparator
010	External Interrupt Request 0
011	Timer/Counter0 Compare Match A
100	Timer/Counter0 Overflow
101	Timer/Counter1 Compare Match B
110	Timer/Counter1 Overflow
111	Timer/Counter1 Capture Event

Πίνακας 3.7: ADC Auto Trigger Source

Καταχωρητές δεδομένων του ADC

Το αποτέλεσμα του ADC έχει μήκος 10-bit και παρουσιάζεται στους καταχωρητές δεδομένων ADCH (high byte) και ADCL (low byte). Από προεπιλογή, το αποτέλεσμα παρουσιάζεται δεξιά προσαρμοσμένο(right adjusted), αλλά μπορεί προαιρετικά να τεθεί η σημαία ADLAR του καταχωρητή ADMUX και να παρουσιαστεί αριστερά προσαρμοσμένο (left adjusted).

Κατά την ανάγνωση του ADCL, οι καταχωρητές δεδομένων του ADC δεν ενημερώνονται μέχρι να διαβαστεί το ADCH. Εάν το αποτέλεσμα είναι αριστερά προσαρμοσμένο(Left adjusted) και δεν απαιτείται ακρίβεια μεγαλύτερη από 8 bit, αρκεί να διαβαστεί ο καταχωρητής ADCH. Σε διαφορετική περίπτωση, πρέπει πρώτα να διαβαστεί το ADCL και μετά το ADCH.

Οι καταχωρητές δεδομένων του ADC, ADCL και ADCH έχουν διευθύνσεις 0x78 και 0x79 αντίστοιχα. Παρουσιάζονται στο επόμενο σχήμα:

ADC Data Register Low and High Byte (ADLAR=0, Right adjusted)

Bit	15	14	13	12	11	10	9	8
	[]	[]	[]	[]	[]	[]	ADC9	ADC8
Access							R	R
Reset							0	0
Bit	7	6	5	4	3	2	1	0
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

ADC Data Register Low and High Byte (ADLAR=1, Left adjusted)

Bit	15	14	13	12	11	10	9	8
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0
Bit	7	6	5	4	3	2	1	0
	ADC1	ADC0	[]	[]	[]	[]	[]	[]
Access	R	R						
Reset	0	0						

Σχήμα 4.6: Καταχωρητές ADCL και ADCH

DIDR0 (Διεύθυνση: 0x7E), Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Access	R/W							
Reset	0	0	0	0	0	0	0	0

Σχήμα 3.7: Καταχωρητής DIDR0

Όταν ένα bit αυτού του καταχωρητή τίθεται, τότε ο απομονωτής του αντίστοιχου ακροδέκτη του ADC απενεργοποιείται. Εάν ένα αναλογικό σήμα εφαρμόζεται σε έναν από τους ακροδέκτες ADC[7:0] και η ψηφιακή είσοδος από αυτόν τον ακροδέκτη δεν χρειάζεται τότε το αντίστοιχο bit ADCxD θα πρέπει να τεθεί για να μειωθεί η κατανάλωση ενέργειας στον απομονωτή της ψηφιακής εισόδου.

Παραγωγή αναλογικών τάσεων στο ntuAboard_G1

Η κάρτα ntuAboard_G1 διαθέτει κυκλώματα για την παραγωγή αναλογικών τάσεων, οι οποίες μπορούν να συνδεθούν στις εισόδους 0 έως 3 του ADC μετατροπέα. Ποιο συγκεκριμένα η κάρτα ntuAboard_G1 διαθέτει 4 ποτενσιόμετρα, 3 αναλογικά φίλτρα τα οποία παράγουν μεταβαλλόμενες DC τάσεις από τις PWM εξόδους των χρονιστών και ένα μετατροπέα ψηφιακού σήματος σε αναλογικό(DAC).

Τα κυκλώματα που αναφέρθηκαν απεικονίζονται στο παρακάτω σχήμα:



Σχήμα 3.4: Κυκλώματα για την παραγωγή αναλογικών τάσεων στην κάρτα ntuAboard_G1.

Η σύνδεση καθεμιάς από τις πρώτες 4 αναλογικές εισόδους του ADC με τις αναλογικές τάσεις που παράγονται στην κάρτα ntuAboard_G1 μπορεί να επιτευχθεί χρησιμοποιώντας βραχυκυκλωτήρες:

- Η αναλογική είσοδος 0 του ADC μπορεί να συνδεθεί είτε με το ποτενσιόμετρο POT1 είτε με το αναλογικό φίλτρο PD6_PWM.
- Η αναλογική είσοδος 1 του ADC μπορεί να συνδεθεί είτε με το ποτενσιόμετρο POT2 είτε με το αναλογικό φίλτρο PB1_PWM.
- Η αναλογική είσοδος 2 του ADC μπορεί να συνδεθεί είτε με το ποτενσιόμετρο POT3 είτε με το αναλογικό φίλτρο PD3_PWM.
- Η αναλογική είσοδος 3 του ADC μπορεί να συνδεθεί είτε με το ποτενσιόμετρο POT4 είτε με την έξοδο του DAC.

Να σημειωθεί ότι για να μπορέσει να λειτουργήσει το αναλογικό φίλτρο στο PD6_PWM πρέπει να συνδεθεί ένας βραχυκυκλωτήρας στον κονέκτορα J3. Παρομοίως το φίλτρο PB1_PWM απαιτεί ένα βραχυκυκλωτήρα στον κονέκτορα J5 και το φίλτρο PD3_PWM απαιτεί ένα βραχυκυκλωτήρα στον κονέκτορα J7.

Για παράδειγμα στο σχήμα 4.8 η αναλογική είσοδος A0 έχει συνδεθεί στο ποτενσιόμετρο POT1, η αναλογική είσοδος A1 έχει συνδεθεί στο PB1_PWM φίλτρο και οι αναλογικές είσοδοι A2 και A3 παραμένουν ασύνδετες.

Για να λειτουργήσουν σωστά οι αναλογικές είσοδοι της PORTC πρέπει να αποσυνδεθεί η θύρα αυτή από τα led δια μέσω του DIP switch.

Παράδειγμα 3.2

Στο παρακάτω παράδειγμα ο ADC έχει ρυθμιστεί έτσι ώστε να διαβάζει συνεχώς την αναλογική είσοδο 0. Το αποτέλεσμα της μετατροπής είναι αριστερά προσαρμοσμένο (Left adjusted) και τα 8 σημαντικότερα bit απεικονίζονται στα Led του PORTD.

Η τάση αναφοράς έχει ρυθμιστεί στα 5 volt.

Η συχνότητα του ADC ισούται με 125 KHz και προκύπτει από τη διαίρεση της συχνότητας λειτουργίας του μικροελεγκτή(16MHz) με το συντελεστή 128.

```

.include "m328PBdef.inc" ;ATmega328P microcontroller definitions

.def temp = r16
.def ADC_L = r21
.def ADC_H = r22

.org 0x00
    jmp reset

.org 0x2A ;ADC Conversion Complete Interrupt
    reti

reset:
    ldi temp, high(RAMEND)
    out SPH,temp
    ldi temp, low(RAMEND)
    out SPL,temp

    ldi temp, 0xFF
    out DDRD, temp ;Set PORTD as output

    ldi temp, 0x00
    out DDRC, temp ;Set PORTC as input

; REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0000 => select ADC0(pin PC0),
; ADLAR=1 => Left adjust the ADC result
ldi temp, 0b01100000 ;
sts ADMUX, temp

; ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
; ADIE=0 => disable adc interrupt, ADPS[2:0]=111 => fADC=16MHz/128=125KHz
ldi temp, 0b10000111
sts ADCSRA, temp

Start_conv:
    lds temp, ADCSRA ;
    ori temp, (1<<ADSC) ; Set ADSC flag of ADCSRA
    sts ADCSRA, temp ;

wait_adc:
    lds temp, ADCSRA ;
    sbrc temp,ADSC ; Wait until ADSC flag of ADCSRA becomes 0
    rjmp wait_adc ;

    lds ADC_L,ADCL ; Read ADC result(Left adjusted)
    lds ADC_H,ADCH ;
    out PORTD, ADC_H ; Output ADCH to PORTD

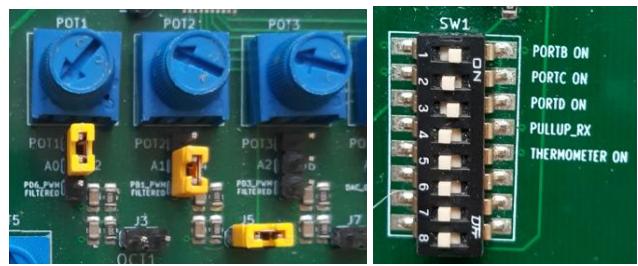
    rjmp Start_conv

```

Τα ζητούμενα της 3^{ης} εργαστηριακής άσκησης

Για την υλοποίηση των επομένων εργασιών συνδέστε την είσοδο A0 του ADC με το POT1, την είσοδο A1 του ADC με το αναλογικό φίλτρο PB1_PWM, και γεφυρώστε το J5 όπως φαίνεται στο διπλανό σχήμα.

Το PORTC περιλαμβάνει αναλογικές εισόδους του ADC, οπότε όταν γίνεται χρήση του ADC τότε τα led που ελέγχονται από το PORTC πρέπει να αποσυνδεθούν, κάνοντας χρήση των dip switches SW1.



Ζήτημα 3.1

Να δημιουργηθεί κώδικας σε γλώσσα assembly, ο οποίος να αρχικοποιεί τον TMR1A σε λειτουργία 8-bit και να παράγει μία PWM κυματομορφή στον ακροδέκτη PB1 με συχνότητα 62500 Hz. Θεωρήστε ότι BOTTOM=0. Στον ακροδέκτη PB1 είναι συνδεδεμένο ένα LED και η φωτεινότητα του μεταβάλλεται εάν μεταβληθεί το Duty Cycle της PWM κυματομορφής. Αρχικά το Duty Cycle ρυθμίζεται σε 50% αποθηκεύεται σε μία μεταβλητή με όνομα DC_VALUE. Στη συνέχεια κάθε φορά που πιέζεται το μπουτόν PB4 το Duty Cycle αυξάνεται κατά 6%. Όταν το Duty Cycle φτάσει τη μέγιστη τιμή 98% η πίεση του μπουτόν PB4 δεν το αυξάνει περεταίρω. Κάθε φορά που πιέζεται το μπουτόν PB5 το Duty Cycle μειώνεται κατά 6%. Όταν το Duty Cycle φτάσει τη ελάχιστη τιμή 2% η πίεση του μπουτόν PB5 δεν το μειώνει περεταίρω. Οι τιμές που πρέπει να πάρει ο καταχωρητής OCR1A, για τις διάφορες τιμές του Duty Cycle, να έχουν υπολογιστεί εκ των προτέρων και να έχουν τοποθετηθεί σε ένα πίνακα στη μνήμη του μικροελεγκτή, έτσι ώστε να μη χρειάζεται να υπολογιστούν κατά τη διάρκεια εκτέλεσης του κώδικα.

Ζήτημα 3.2

Να γραφτεί ξανά ο κώδικας του ζητήματος 3.1 σε γλώσσα C, ο οποίος θα παράγει PWM έξοδο στον ακροδέκτη PB1, με συχνότητα 62500 Hz.

Κάθε 100 mSec (μικρές αποκλίσεις είναι αποδεκτές), ο ADC(ADCH: ADCL) θα διαβάζει την DC τάση που παράγεται στην έξοδο του αναλογικού φίλτρου PB1_PWM με ακρίβεια 10 bit.

Να υπολογίζεται η μέση τιμή του ADC αθροίζοντας 16 διαδοχικές μετρήσεις και διαιρώντας δια 16. Προκειμένου να επιτευχθεί η ταχύτερη εκτέλεση του κώδικα, η διαίρεση θα υλοποιείται με ολίσθηση των bit του αθροίσματος κατά 4 θέσεις. Ανάλογα με την τιμή μέτρησης του ADC (ADCH: ADCL) θα ανάβει ένα από τα led που είναι συνδεδεμένα στη θύρα PORTD σύμφωνα με τον παρακάτω πίνακα:

Τιμή μέτρησης του ADC(ADCvalue)	LED ON
$0 \leq \text{ADCvalue} \leq 200$	PD0
$200 \leq \text{ADCvalue} \leq 400$	PD1
$400 \leq \text{ADCvalue} \leq 600$	PD2
$600 \leq \text{ADCvalue} \leq 800$	PD3
$800 \leq \text{ADCvalue}$	PD4

Παρατήρηση: Είναι φυσιολογικό να τρεμοπαίζουν τα led στα σημεία μετάβασης.

Ζήτημα 3.3

Να γραφτεί ξανά ο κώδικας του ζητήματος 3.1 σε γλώσσα C.

Επιπλέον, για τη μεταβολή της φωτεινότητας του led θα υπάρχουν δύο τρόποι λειτουργίας, το mode1 και το mode2. Στο mode1 η φωτεινότητα(Duty Cycle) θα αποθηκεύεται σε μια μεταβλητή με όνομα DC_VALUE και το πάτημα του μπουτόν PB4 θα αυξάνει τη φωτεινότητα ενώ το πάτημα του μπουτόν PB5 θα μειώνει τη φωτεινότητα.

Στο mode2 η φωτεινότητα(Duty Cycle) θα ελέγχεται από το ποτενσιόμετρο POT1.

Εάν πατηθεί το μπουτόν PD0 τότε θα επιλέγεται το mode1.

Εάν πατηθεί το μπουτόν PD1 τότε θα επιλέγεται το mode2.



Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΑΘΗΝΑ, 31/10/2025

4^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικρούπολογιστών"
Χρήση Οθόνης 2x16 Χαρακτήρων στον AVR

Εξέταση – Επίδειξη: Παρασκευή 7/11/2025
Προθεσμία για παράδοση Έκθεσης: Τρίτη 11/11/2025 (23:59)

Αλφαριθμητική Οθόνη 2x16 Χαρακτήρων

Η οθόνη είναι υγρού κρυστάλλου (LCD) με χαμηλή κατανάλωση και έχει ενσωματωμένο ένα μικροελεγκτή. Ο ενσωματωμένος μικροελεγκτής εμπεριέχει δύο μνήμες για τη δημιουργία χαρακτήρων: την CG-ROM (Character Generation ROM) και την CG-RAM (Character Generation RAM). Επίσης έχει μία μνήμη RAM στην οποία κρατούνται προσωρινά οι χαρακτήρες που απεικονίζονται κάθε φορά στην οθόνη την DD_RAM (Display Data RAM).

Ακροδέκτες της οθόνης

Στη συνέχεια παρατίθενται οι ακροδέκτες της οθόνης και οι αντίστοιχες λειτουργίες τους:

DB[0:7]	Τρικατάστατος αμφιδρομος διάδρομος δεδομένων 8-γραμμών, με τον οποίο η οθόνη ανταλλάσσει δεδομένα με τον εξωτερικό μικροεπεξεργαστή. Υπάρχει δυνατότητα χρήσης μόνο 4 γραμμών, οπότε η ανταλλαγή των δεδομένων γίνεται σε δύο φάσεις, δια μέσω των ακροδεκτών DB[7:4] ενώ οι ακροδέκτες DB[0:3] δεν χρησιμοποιούνται. Τα 4 περισσότερο σημαντικά bit πρέπει να μεταφέρονται πρώτα.
E'	Σήμα που ενεργοποιεί την οθόνη. Η ενεργοποίηση γίνεται Αν E'=0.
R/W'	Σήμα που καθορίζει αν πρόκειται να γίνει εγγραφή ή διάβασμα από την οθόνη (R/W'=0 εγγραφή, R/W'=1 διάβασμα).
RS	Το σήμα RS καθορίζει αν το περιεχόμενο των DB[0:7] είναι διεύθυνση/εντολή ή αν είναι δεδομένα που πρόκειται να διαβαστούν ή να εγγραφούν (RS =0 διεύθυνση/εντολή, RS=1 δεδομένα).
V0	Ρυθμίζει τη φωτεινότητα της οθόνης.
VDD	Συνδέεται με την τάση τροφοδοσίας (3,3 volt).
VSS	Συνδέεται με τη γη (0 volt).

Καταχωρητές

Ο εσωτερικός μικροελεγκτής της LCD οθόνης ενσωματώνει δύο καταχωρητές των 8-bit, τον καταχωρητή εντολών (instruction register -IR) και τον καταχωρητή δεδομένων (data register- DR).

Ο καταχωρητής IR αποθηκεύει τον κώδικα των εντολών, ή τη διεύθυνση μιας θέσης της μνήμης DD-RAM ή της μνήμης CG-RAM. Στον καταχωρητή IR γίνεται μόνο εγγραφή και όχι ανάγνωση από τον εξωτερικό μικροελεγκτή. Ο καταχωρητής DR αποθηκεύει προσωρινά τα δεδομένα που διαβάζονται η γράφονται στην DD-RAM ή στην CG-RAM.

Σημαία απασχόλησης (Busy Flag-BF)

Όταν η σημαία απασχόλησης είναι 1, η οθόνη είναι σε κατάσταση εσωτερικής λειτουργίας και δεν γίνεται δεκτή καμία άλλη εντολή. Όταν RS = 0 και R/W' = 1, η σημαία απασχόλησης εμφανίζεται στην έξοδο DB7. Η επόμενη εντολή πρέπει να 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 DD-RAM και CG-RAM. Όταν μια διεύθυνση μιας εντολής εγγράφεται στον IR, η διεύθυνση στέλνεται από τον IR στον AC.

Μετά την εγγραφή ή την ανάγνωση από την DD-RAM ή την CG-RAM, η AC αυτόματα αυξάνεται κατά 1 (ή ελαττώνεται κατά 1). Το περιεχόμενο του καταχωρητή AC δίνεται στη έξοδο μέσω των DB[0:6] όταν RS = 0 και R/W' = 1.

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 DD-RAM or CG-RAM)
1	1	DR read as an internal operation (DD-RAM or CG-RAM to DR)

Πίνακας 4.1: Επιλογή Καταχωρητή

Εσωτερική μνήμη DD-RAM

Η DD-RAM αποθηκεύει τα δεδομένα που πρόκειται να απεικονιστούν στην οθόνη. Η χωρητικότητά της είναι 80 byte. Ο κώδικας της διεύθυνσης μιας θέσης μνήμης DD-RAM είναι των επτά Bit. Οι χαρακτήρες που οι κώδικες τους βρίσκονται στις θέσεις 0x00 έως 0x0F της DD-RAM τοποθετούνται(απεικονίζονται) στη γραμμή 1 της οθόνης, ενώ οι κώδικες που βρίσκονται στις θέσεις 0x40 έως 0x4F απεικονίζονται στη γραμμή 2 όπως φαίνονται στον παρακάτω πίνακα:

Γραμμή_1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Γραμμή_2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Πίνακας 4.2: Απεικόνιση χαρακτήρων

Αν πρέπει να απεικονιστούν και άλλοι χαρακτήρες που οι κώδικες τους είναι αποθηκευμένοι στη DD-RAM, εκτός από τους 32 χαρακτήρες που αναφέρθηκαν πιο πάνω, τότε πρέπει να γίνει ολίσθηση στο περιεχόμενο της οθόνης δεξιά ή αριστερά.

Εσωτερική μνήμη CG-ROM

Η CG-ROM χρησιμοποιείται για τη δημιουργία εικόνων χαρακτήρων 5×8 κουκίδων από κωδικούς χαρακτήρων των 8 bit. Η αντιστοιχία αριθμών των 8 bit και χαρακτήρων φαίνεται στον ακόλουθο πίνακα.

Upper 4 bit Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHHL	HHHH
CG RAM (1)	‡	@@P “PSeéá	·	¶P3T										
CG RAM (2)	■■!	1A0a9Oeí	·	Jtyv										
CG RAM (3)	?”2B	Rb r@Eš “o@Sx												
CG RAM (4)	‡#3C	Scs.88G “PMeψ												
CG RAM (5)	‡\$4D	Tdt.88t “ePZo												
CG RAM (6)	‡\$E	Ueu.88t “†dmw												
CG RAM (7)	‡\$F	Ufv.88t “G@p4u489w												
CG RAM (8)	‡\$G	Uew.88t “x@Pz@h u												
CG RAM (1)	‡\$H	Xhx.88t “S k												
CG RAM (2)	‡\$I	Yi w@o i SPTλ@												
CG RAM (3)	‡\$J	Zj z@O@2 T@μP												
CG RAM (4)	‡\$K	Kk C1R@ <L@V@												
CG RAM (5)	‡\$L	>1 I 3R@ >J@G@												
CG RAM (6)	‡\$M	Mm3 3@S@ “Ψπ=												
CG RAM (7)	‡\$N	Nn^ A@S@ T@S@p@												
CG RAM (8)	‡\$O	OoA@A@φ “@o@t@												

Πίνακας 4.3: Αντιστοιχία αριθμών των 8 bit και χαρακτήρων.

Εντολές της οθόνης

Επειδή στην κάρτα ntuAboard_G1 δεν υπάρχει δυνατότητα ανάγνωσης της σημαίας BUSY FLAG για να γνωρίζουμε πότε η οθόνη είναι έτοιμη να δεχτεί νέα εντολή πρέπει να εισάγουμε καθυστέρηση μεταξύ των διαδοχικών εντολών. Ο χρόνος που χρειάζεται για την εκτέλεση κάθε εντολής είναι μεταβλητός και εξαρτάται από την κάθε εντολή. Το σύνολο εντολών της οθόνης περιέχεται στον ακόλουθο πίνακα.

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 μ s
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 μ s
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 μ s
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:5x11 dots/5x8 dots)	39 μ s
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0		Set CGRAM address in address counter.	39 μ s
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0		Set DDRAM address in address counter.	39 μ s
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 μ s
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0		Write data into internal RAM (DDRAM/CGRAM).	43 μ s
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0		Read data from internal RAM (DDRAM/CGRAM).	43 μ s

* ”—” : don't care

Πίνακας 4.4: Το σύνολο εντολών της οθόνης

Παρατήρηση: Πειραματικά έχει φανεί ότι οι οθόνες που χρησιμοποιούνται στην κάρτα ntuAboard_G1 δεν μπορούν να ανταποκριθούν στους χρόνους που φαίνονται στον παραπάνω πίνακα. Για το σκοπό αυτό, στα τμήματα του κώδικα που ακολουθούν, έχουν χρησιμοποιηθεί μεγαλύτερες καθυστερήσεις. Αυτό πρέπει να ληφθεί υπόψιν και στον κώδικα που θα δημιουργήσετε.

Στη συνέχεια δίνεται η επεξήγηση των παραμέτρων που χρησιμοποιούνται στις παραπάνω εντολές.

To ACx, x=0 έως 6, συμβολίζει ένα bit του address counter.

To Dx, x=0 έως 7, συμβολίζει ένα bit δεδομένων.

To BF συμβολίζει το busy flag.

Στον πίνακα που ακολουθεί παρουσιάζεται η ρύθμιση του τρόπου λειτουργίας της οθόνης στην είσοδο ή έξοδο δεδομένων:

S	I/D	Λειτουργία
0	0	Η διεύθυνση της DD-RAM ή της CG-RAM μειώνεται κατά 1 όταν γράφουν δεδομένα. Ο κέρσορας μετακινείται προς τα αριστερά.
0	1	Η διεύθυνση της DD-RAM ή της CG-RAM αυξάνεται κατά 1 όταν γράφουν δεδομένα. Ο κέρσορας μετακινείται προς τα δεξιά.
1	0	Το περιεχόμενο της οθόνης ολισθαίνει προς τα δεξιά όταν γράφουμε στην DD-RAM (όχι όταν διαβάζουμε). Η θέση του κέρσορα δεν μεταβάλλεται.
1	1	Το περιεχόμενο της οθόνης ολισθαίνει προς τα αριστερά όταν γράφουμε στην DD-RAM (όχι όταν διαβάζουμε). Η θέση του κέρσορα δεν μεταβάλλεται.

Πίνακας 4.5: Ρύθμιση του τρόπου λειτουργίας της οθόνης στην είσοδο ή έξοδο δεδομένων

Στον πίνακα που ακολουθεί παρουσιάζεται η ρύθμιση του τρόπου ολίσθησης της οθόνης:

S/C	R/L	Λειτουργία
0	0	Η θέση του κέρσορα μετακινείται προς τα αριστερά. (Ο AC μειώνεται κατά 1).
0	1	Η θέση του κέρσορα μετακινείται προς τα δεξιά. (Ο AC αυξάνεται κατά 1).
1	0	Το περιεχόμενο της οθόνης ολισθαίνει προς τα αριστερά μαζί με τον κέρσορα. (Ο AC δεν μεταβάλλεται).
1	1	Το περιεχόμενο της οθόνης ολισθαίνει προς τα δεξιά μαζί με τον κέρσορα. (Ο AC δεν μεταβάλλεται).

Πίνακας 4.6: Ρύθμιση του τρόπου ολίσθησης της οθόνης

Αν D=1 τότε η μονάδα απεικόνισης της οθόνης είναι ανοιχτή. Αν D=0 τότε η μονάδα απεικόνισης της οθόνης είναι κλειστή. Και στις δύο περιπτώσεις τα δεδομένα της DD-RAM μένουν αμετάβλητα.

Αν C=1 τότε ο κέρσορας απεικονίζεται στην οθόνη. Αν C=0 τότε ο κέρσορας δεν απεικονίζεται στην οθόνη.

Αν B=1 τότε ο χαρακτήρας πάνω από τον κέρσορα αναβοσβήνει. Αν B=0 τότε ο χαρακτήρας δεν αναβοσβήνει.

Αν DL=1 τότε το μήκος των δεδομένων είναι 8 bit. Αν DL=0 τότε το μήκος των δεδομένων είναι 4 bit.

Στον παρακάτω πίνακα παρουσιάζονται οι κωδικοί των βασικών εντολών της οθόνης:

A/A	Hex Code	Command to LCD instruction Register
1	1	Clear display screen
2	2	Return home
3	4	Decrement cursor (shift cursor to left)
4	6	Increment cursor (shift cursor to right)
5	5	Shift display right
6	7	Shift display left
7	8	Display off, cursor off
8	0A	Display off, cursor on
9	0C	Display on, cursor off
10	0E	Display on, cursor blinking
11	0F	Display on, cursor blinking
12	10	Shift cursor position to left
13	14	Shift the cursor position to the right
14	18	Shift the entire display to the left
15	1C	Shift the entire display to the right
16	80	Force cursor to the beginning (1st line)
17	C0	Force cursor to the beginning (2nd line)
18	38	2 lines and 5x7 matrix

Πίνακας 4.7: Κωδικοί των βασικών εντολών της οθόνης

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

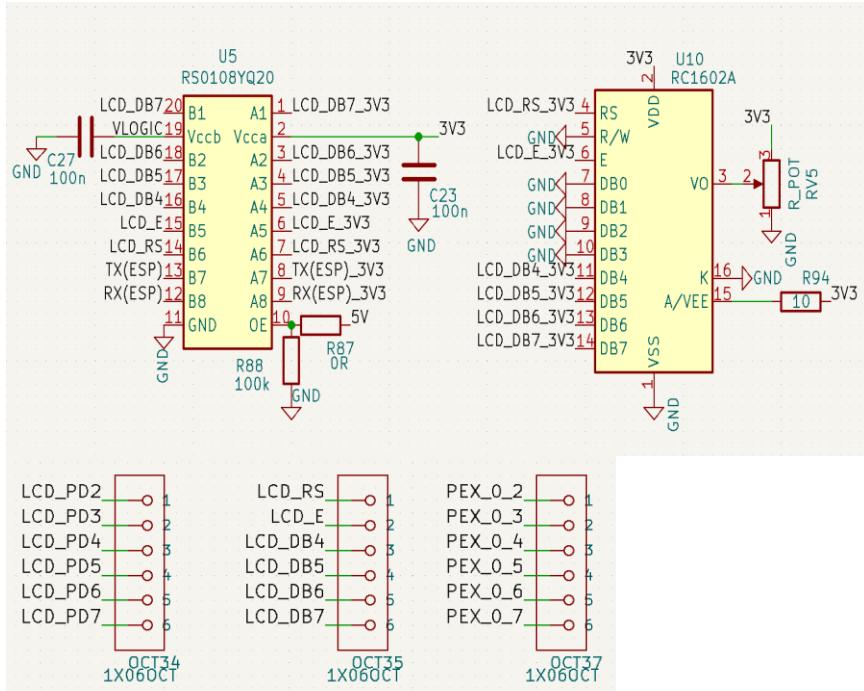
Ta=25°C, VDD=5,0V						
Item	Symbol	Min	Typ	Max	Unit	
Enable cycle time	T _C	1200	—	—	ns	
Enable pulse width	T _{PW}	140	—	—	ns	
Enable rise/fall time	T _R ,T _F	—	—	25	ns	
Address set-up time (RS, R/W to E)	t _{AS}	0	—	—	ns	
Address hold time	t _{AH}	10	—	—	ns	
Data set-up time	t _{DSW}	40	—	—	ns	
Data hold time	t _H	10	—	—	ns	

Πίνακας 4.8: Χρονισμός εντολών οθόνης

Χρήση της οθόνης LCD 2x16 της κάρτας ntuAboard_G1.

Η κάρτα ntuAboard_G1 διαθέτει μία LCD οθόνη 2x16 χαρακτήρων. Μεταξύ της οθόνης και του μικροελεγκτή παρεμβάλλεται το ολοκληρωμένο RS0108YQ20 (8-Bit Bidirectional Voltage Level Translator) για προσαρμογή του επιπέδου της τάσης του μικροελεγκτή (5 Volt) με το επίπεδο της τάσης της οθόνης (3.3 Volt).

Η συνδεσμολογία της οθόνης LCD με τον Μικροελεγκτή ATmega328PB φαίνεται στο παρακάτω σχήμα:



Σχήμα 4.1: Συνδεσμολογία της οθόνης LCD

Κάνοντας χρήση των κονεκτόρων OCT34, OCT35 και OCT37 μπορούμε να επιλέξουμε εάν οι ακροδέκτες ελέγχου της LCD οθόνης (OCT35) θα συνδέθουν στο PORTD του ATmega328PB (OCT34) ή στο PORT EXPANDER (OCT37). Ανάλογα με τη σύνδεση, οι ακροδέκτες [LCD_PD2 : LCD_PD7] ή οι ακροδέκτες [PEX_0_7 : PEX_0_2] πρέπει να είναι ρυθμισμένοι για έξοδο. Η επικοινωνία μεταξύ οθόνης και του μικροελεγκτή γίνεται με λέξεις των 4 bit.

Κώδικας επικοινωνίας μεταξύ μικροελεγκτή και οθόνης

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

write_2_nibbles:

```

push r24 ; save r24(LCD_Data)
in r25 ,PIND ; read PIND
andi r25 ,0x0f ;
andi r24 ,0xf0 ; r24[3:0] Holds previous PORTD[3:0]
add r24 ,r25 ; r24[7:4] <- LCD_Data_High_Byt
out PORTD ,r24 ;

```

```

sbi PORTD ,PD3           ; Enable Pulse
nop
nop
cbi PORTD ,PD3

pop r24                  ; Recover r24(LCD_Data)
swap r24
andi r24 ,0xf0          ; r24[3:0] Holds previous PORTD[3:0]
add r24 ,r25              ; r24[7:4] <-- LCD_Data_Low_Byte
out PORTD ,r24

sbi PORTD ,PD3           ; Enable Pulse
nop
nop
cbi PORTD ,PD3

ret

```

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

Poutrίνα: lcd_data

Αποστολή ενός byte δεδομένων στον ελεγκτή της οθόνης lcd. Ο ελεγκτής πρέπει να βρίσκεται σε 4 bit mode. Το byte που μεταδίδεται είναι αποθηκευμένο στον καταχωρητή r24

```

lcd_data:
    sbi PORTD ,PD2          ; LCD_RS=1(PD2=1), Data
    rcall write_2_nibbles    ; send data
    ldi r24 ,250             ;
    ldi r25 ,0                ; Wait 250uSec
    rcall wait_usec
    ret

```

Poutrίνα: lcd_command

Αποστολή μιας εντολής στον ελεγκτή της οθόνης lcd.

Ο ελεγκτής πρέπει να βρίσκεται σε 4 bit mode.

Η εντολή που μεταδίδεται είναι αποθηκευμένη στον καταχωρητή r24

```

lcd_command:
    cbi PORTD ,PD2          ; LCD_RS=0(PD2=0), Instruction
    rcall write_2_nibbles    ; send Instruction
    ldi r24 ,250             ;
    ldi r25 ,0                ; Wait 250uSec
    rcall wait_usec
    ret

```

Poutrίνα: lcd_clear_display

Καθαρισμός της οθόνης LCD :

```

lcd_clear_display:

```

```

ldi r24 ,0x01           ; clear display command
rcall lcd_command

ldi r24 ,low(5)          ;
ldi r25 ,high(5)         ; Wait 5 mSec
rcall wait_msec          ;

ret

```

Για να μπορεί να χρησιμοποιηθεί οθόνη πρέπει να αρχικοποιηθεί στην επιθυμητή κατάσταση. Όταν η οθόνη τροφοδοτείται με ρεύμα για πρώτη φορά πραγματοποιείται μια εσωτερική αρχικοποίηση και για αυτό απαιτείται χρονική καθυστέρηση 200 ms πριν την τελική αρχικοποίηση.

Κάθε φορά που προγραμματίζουμε τον Μικροελεγκτή, ξεκινάει η εκτέλεση του κώδικα από την αρχή. Η οθόνη όμως βρίσκεται στην κατάσταση που την αρχίσαμε την προηγούμενη φορά. Ο κώδικας που θα κάνει την αρχικοποίηση της οθόνης δεν γνωρίζει αν ο ελεγκτής της οθόνης βρίσκεται σε 8 bit mode ή σε 4 bit mode. Για το λόγο αυτό αρχικά στέλνεται 3 φορές η εντολή 0x30 (function set) για 8 bit mode. 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. Στη συνέχεια στέλνεται η εντολή 0x20 για να οδηγηθεί η οθόνη σε 4 bit mode. Μόλις είμαστε βέβαιοι για την μορφή που πρέπει να στέλνουμε τις εντολές μπορούμε να προχωρήσουμε με την αρχικοποίηση.

Poντίνα: lcd_init

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 DD-RAM address auto increment
;SH = 0   shift of entire display off

```

lcd_init:

```

ldi r24 ,low(200)          ;
ldi r25 ,high(200)         ; Wait 200 mSec
rcall wait_msec            ;

ldi r24 ,0x30               ; command to switch to 8 bit mode
out PORTD ,r24              ;
sbi PORTD ,PD3              ; Enable Pulse
nop
nop
cbi PORTD ,PD3
ldi r24 ,250                 ;
ldi r25 ,0                   ; Wait 250uSec
rcall wait_usec              ;

ldi r24 ,0x30               ; command to switch to 8 bit mode
out PORTD ,r24              ;

```

```

sbi PORTD ,PD3           ; Enable Pulse
nop
nop
cbi PORTD ,PD3
ldi r24 ,250
ldi r25 ,0
rcall wait_usec
;

ldi r24 ,0x30           ; command to switch to 8 bit mode
out PORTD ,r24
sbi PORTD ,PD3
nop
nop
cbi PORTD ,PD3
ldi r24 ,250
ldi r25 ,0
rcall wait_usec
;

ldi r24 ,0x20           ; command to switch to 4 bit mode
out PORTD ,r24
sbi PORTD ,PD3
nop
nop
cbi PORTD ,PD3
ldi r24 ,250
ldi r25 ,0
rcall wait_usec
;

ldi r24 ,0x28           ; 5x8 dots, 2 lines
rcall lcd_command

ldi r24 ,0x0c           ; display on, cursor off
rcall lcd_command
rcall lcd_clear_display

ldi r24 ,0x06           ; Increase address, no display shift
rcall lcd_command
;
ret

```

Παράδειγμα

Ο κώδικας που ακολουθεί προξενεί επαναληπτικά:

Καθαρισμό της οθόνης - Καθυστέρηση 1 sec - Απεικόνιση του χαρακτήρα ‘A’ στην οθόνη - Καθυστέρηση 1 sec.

```
.include "m328PBdef.inc" ;ATmega328P microcontroller definition
```

```
.
.equ PD0=0
.equ PD1=1
.equ PD2=2
.equ PD3=3
.equ PD4=4
.equ PD5=5
```

```

.equ PD6=6
.equ PD7=7

main:
    ldi r24, low(RAMEND)
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24           ; init stack pointer

    ser r24
    out DDRD, r24          ; set PORTD as output

    clr r24

    rcall lcd_init

    ldi r24, low(100)
    ldi r25, high(100)      ; delay 100 mS
    rcall wait_msec

main1:
    rcall lcd_clear_display

    ldi r24, low(1000)
    ldi r25, high(1000)
    rcall wait_msec          ; delay 1 Sec

    ldi r24, 'A'
    call lcd_data

    ldi r24, low(1000)
    ldi r25, high(1000)
    rcall wait_msec          ; delay 1 Sec
    jmp main1

```

Παραδείγματα ρουτινών χρονοκαθυστερήσεων που χρησιμοποιούνται στα τμήματα κώδικα παραπάνω:

```

wait_msec:

    push r24           ; 2 cycles
    push r25           ; 2 cycles
    ldi r24, low(999)   ; 1 cycle
    ldi r25, high(999)  ; 1 cycle
    rcall wait_usec    ; 998.375 usec
    pop r25            ; 2 cycles
    pop r24            ; 2 cycles
    nop                ; 1 cycle
    nop                ; 1 cycle

    sbiw r24, 1 ; 2 cycles
    brne wait_msec    ; 1 or 2 cycles

    ret                ; 4 cycles

```

```
wait_usec:
    sbiw r24,1          ; 2 cycles (2/16 usec)
    call delay_8cycles   ; 4+8=12 cycles
    brne wait_usec      ; 1 or 2 cycles
    ret
```

```
delay_8cycles:
```

```
    nop
    nop
    nop
    nop
    ret
```

Τα ζητούμενα της 4^{ης} εργαστηριακής άσκησης

Ζήτημα 4.1

Στην κάρτα ntuAboard_G1 να γίνει η σύνδεση του POT4 με την είσοδο A3 του ADC, μέσω κατάλληλου συνδετήρα(Jumper) και να γραφεί πρόγραμμα **σε assembly** για τον ATmega328PB το οποίο θα ξεκινάει μια μετατροπή του ADC κάθε 1Sec. Ο ADC Θα μετατρέπει την τάση που υπάρχει κάθε φορά την είσοδο A1. Η ανάγνωση των δεδομένων του ADC πρέπει να γίνεται μέσα στην ρουτίνα εξυπηρέτησης της διακοπής ολοκλήρωσης μετατροπής του ADC. Η διακοπή αυτή (ADC) μεταφέρει τον έλεγχο στην διεύθυνση **0x02A**, αν είναι ενεργοποιημένη η αντίστοιχη διακοπή (από το bit ADIE του ADCSRA) καθώς και το flag επίτρεψης όλων των διακοπών. Η τιμή μέτρησης του ADC να μετατρέπεται σε τάση και να εκτυπώνονται στην LCD οθόνη, αρχίζοντας κάθε φορά από τον πρώτο χαρακτήρα της πρώτης γραμμής, με ακρίβεια δύο δεκαδικών ψηφίων (δεν χρειάζεται στρογγυλοποίηση).

Η τάση δίνεται από τον τύπο:

$$V_{IN} = \frac{ADC}{1024} V_{REF}$$

Όπου:

- V_{IN} η τάση στην αναλογική είσοδο A3 του μικροελεγκτή.
- ADC η τιμή που διαβάζεται από τον ADC (10bit, από 0-1023)
- V_{REF} η τάση αναφοράς που έχει οριστεί στα 5V.

Ζήτημα 4.2

Να γραφεί σε γλώσσα C, πρόγραμμα για τον ATmega328PB το οποίο θα ξεκινάει μια ADC μετατροπή, όπως στο Ζήτημα 4.1. Δεν θα κάνει χρήση της διακοπής ολοκλήρωσης μετατροπής του ADC και θα περιμένει μέχρι να ολοκληρωθεί η ADC μετατροπή ελέγχοντας το bit ADSC του ADCSRA το οποίο γίνεται 0 μόλις ολοκληρωθεί η μετατροπή (Polling). Θα εκτυπώνει την τάση στην LCD οθόνη με ακρίβεια δύο δεκαδικών ψηφίων.

Ζήτημα 4.3

Να δημιουργηθεί κώδικας **σε C** για την επιτήρηση ενός χώρου όπου υπάρχει αυξημένος κίνδυνος ύπαρξης μονοξειδίου του άνθρακα (CO). Ο αισθητήρας CO είναι συνδεδεμένος στην αναλογική είσοδο A3 του μικροελεγκτή.

Καθ' όλη την διάρκεια πρέπει να διαβάζεται η τιμή του αισθητήρα ανά 100 ms (μικρές αποκλίσεις είναι αποδεκτές) και να εμφανίζεται μια ένδειξη του επιπέδου του αερίου στα LED PB0-PB5.

Αν οποιαδήποτε στιγμή η συγκέντρωση του CO ξεπεράσει τα 75ppm να τυπώνεται στην LCD το μήνυμα GAS DETECTED και να αναβοσβήνουν τα αντίστοιχα LED στα PB0-PB5 αναλόγως του επίπεδου του αερίου.

Το επίπεδο θα πρέπει να συνεχίζει να διαβάζεται (και να εμφανίζεται στα LED) και αν επανέλθει σε επίπεδο χαμηλότερο των 75ppm τα LEDs να σταματούν να αναβοσβήνουν και να τυπώνεται στην LCD το μήνυμα CLEAR. Είναι καλή πρακτική να διατηρείτε τις ρουτίνες εξυπηρέτησης διακοπών όσο το δυνατόν μικρότερες σε χρονική διάρκεια. Παρόλα αυτά αν το πρόγραμμα σας είναι λειτουργικό δεν θα υπάρξει αρνητική επίπτωση στην βαθμολογία.

Για την μέτρηση της συγκέντρωσης του CO χρησιμοποιείται αισθητήρας του τύπου ULPSM-CO-968-001: (https://helios.ntua.gr/pluginfile.php/250446/mod_folder/content/0/ULPSM-CO-968-001.pdf?forcedownload=1), ο οποίος βρίσκεται σε σταθερές συνθήκες θερμοκρασίας (20°C) και υγρασίας (40%). Δίνεται ότι $V_{gas} = 0.1V$ και sensitivity code 129nA/ppm. Για να προσομοιώσετε την έξοδο που θα είχε ο αισθητήρας CO να χρησιμοποιήσετε το ποτενσιόμετρο POT4 στην κάρτα ntuAboard_G1 το οποίο παράγει μια DC τάση στο εύρος 0-5V.



Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΑΘΗΝΑ, 7/11/2025

5^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικρούπολογιστών"
Χρήση εξωτερικών Θυρών Επέκτασης στον AVR

Εξέταση – Επίδειξη: Παρασκευή 14/11/2025
Προθεσμία για παράδοση Έκθεσης: Τρίτη 18/11/2025 (23:59)

Διεπαφή TWI (Two wire Serial Interface)

Το πρωτόκολλο TWI επιτρέπει τη διασύνδεση έως και 127 διαφορετικών συσκευών χρησιμοποιώντας μόνο δύο αμφidρομες γραμμές διαύλου, μία για ρολόι (SCL) και μία για δεδομένα (SDA). Τα μόνα εξωτερικά υλικά που χρειάζονται για την υλοποίηση του διαύλου είναι δύο αντιστάσεις πρόσδεσης (pull-up resistors) για τις γραμμές του διαύλου TWI.

- Όλες οι συσκευές που είναι συνδεδεμένες στο δίαυλο TWI έχουν ατομικές διευθύνσεις.
- Το πρωτόκολλο TWI ενσωματώνει μηχανισμό αποφυγής σύγκρουσης δεδομένων στο δίαυλο.
- Το πρωτόκολλο TWI είναι συμβατό με το πρωτόκολλο I²C της Philips.

Διεπαφές TWI στο ntuAboard_G1

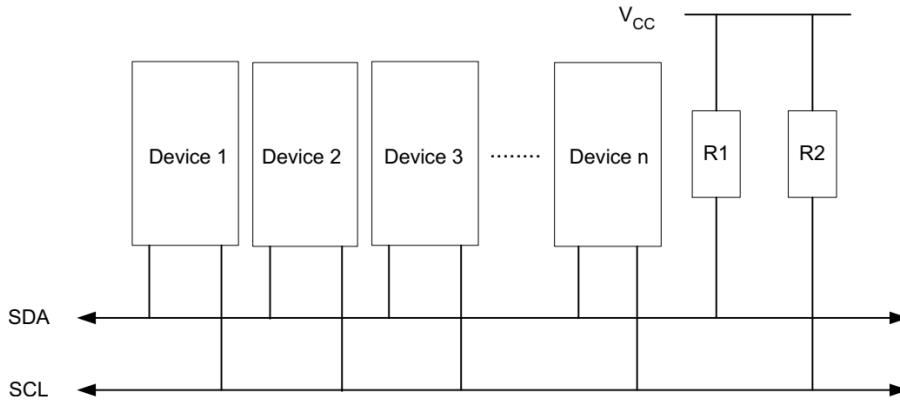
Στο ntuAboard_G1_υπάρχουν δύο διεπαφές TWI, η TWI0 και η TWI1. Οι πληροφορίες που παρουσιάζονται στη συνέχεια αναφέρονται στη διεπαφή TWI0. Ακριβώς τα ίδια ισχύουν και για διεπαφή TWI1.

Οι δύο γραμμές του TWI0 διαύλου συνδέονται στους ακροδέκτες PC4 (SDA) και PC5 (SCL) του ATmega328PB. Οι αντιστάσεις πρόσδεσης (pull-up resistors) έχουν τιμή 10 K_{ohm} και συνδέονται στο δίαυλο με την τοποθέτηση βραχυκυκλωτήρων στους κονέκτορες J12 και J13.



Σχήμα 5.1: Κονέκτορες J12 και J13 για σύνδεση των TWI pull-up resistors

Όταν ο δίαιυλος TWI είναι ενεργός τότε οι ακροδέκτες PC4 (SDA) και PC5 (SCL) δεν μπορούν να χρησιμοποιηθούν για άλλο σκοπό. Στο παρακάτω σχήμα φαίνεται η συνδεσμολογία του διαιύλου TWI



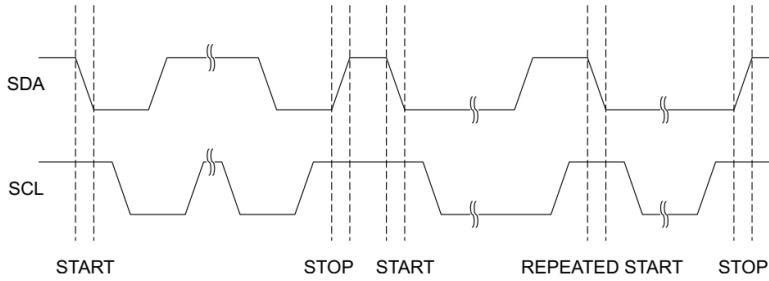
Σχήμα 5.2: Συνδεσμολογία TWI διαιύλου.

Ορολογία διαιύλου TWI:

- **Master:** Η συσκευή που αρχίζει μια μεταφορά, παράγει το σήμα του ρολογιού (SCL) και τερματίζει τη μεταφορά.
- **Slave:** Η συσκευή που καλείται κάθε φορά από τον Master κάνοντας χρήση της ατομικής διεύθυνσης της.
- **Transmitter:** Η συσκευή που στέλνει δεδομένα στο δίαιυλο.
- **Receiver:** Η συσκευή που λαμβάνει δεδομένα από το δίαιυλο.
- **Multi-master:** Η δυνατότητα ύπαρξης πολλών Master στο δίαιυλο, χωρίς απώλεια δεδομένων λόγω σύγκρουσης. Ωστόσο, κάθε χρονική στιγμή, υπάρχει ενεργός μόνο ένας Master στο δίαιυλο.
- **Arbitration:** Η διαδικασία που εγκρίνει κάθε φορά, μόνο έναν Master, να πάρει τον έλεγχο του διαιύλου.
- **Synchronization:** Η διαδικασία που συγχρονίζει τα σήματα ρολογιών που παρέχονται από δύο ή περισσότερους Master.
- **SDA:** Η γραμμή του σήματος δεδομένων.
- **SCL:** Η γραμμή του σήματος ρολογιού

Συνθήκες START και STOP

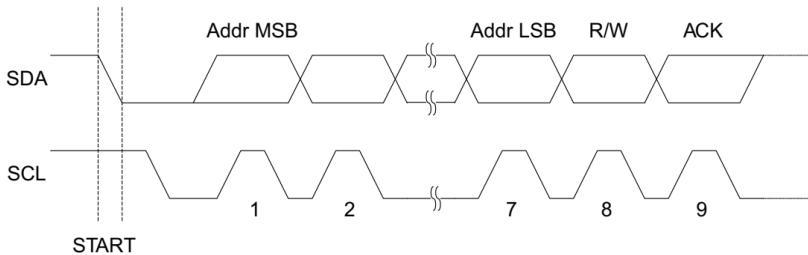
Η μετάδοση ξεκινά όταν ο Master εκδίδει μια συνθήκη START στο δίαιυλο και τερματίζεται όταν ο Master εκδίδει μια συνθήκη STOP. Μεταξύ μιας συνθήκης START και STOP, ο δίαιυλος θεωρείται απασχολημένος και κανένας άλλος Master δεν πρέπει να προσπαθήσει να πάρει τον έλεγχο του διαιύλου. Μια ειδική περίπτωση παρουσιάζεται όταν εκδίδεται μια νέα συνθήκη START μεταξύ μιας συνθήκης START και STOP. Αυτό αναφέρεται ως συνθήκη REPEATED START και χρησιμοποιείται όταν ο Master επιθυμεί να ξεκινήσει μια νέα μεταφορά χωρίς να παραιτηθεί από τον έλεγχο του διαιύλου.



Σχήμα 5.3 Συνθήκες START, REPEATED START, και STOP

Διευθυνσιοδότηση TWI διαύλου

Όλα τα πακέτα διευθύνσεων που μεταδίδονται στον δίαυλο TWI έχουν μήκος εννέα bit, όπως φαίνεται στο παρακάτω σχήμα:

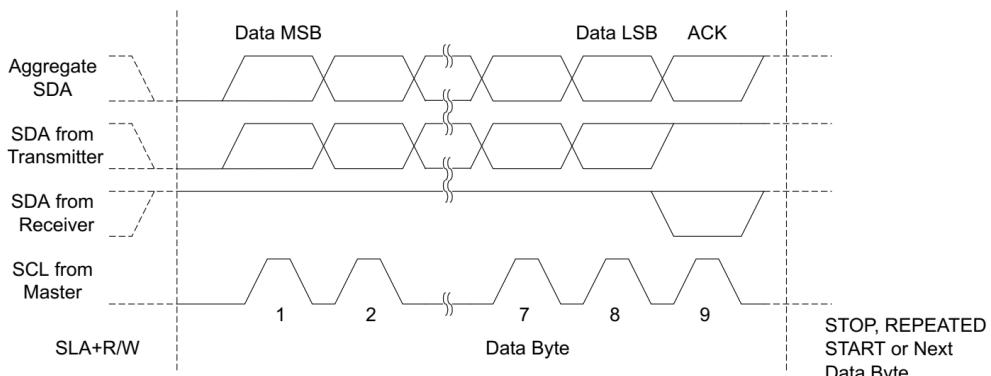


Σχήμα 5.4 Μορφή πακέτου διευθύνσεων

Αποτελούνται από επτά bit διεύθυνσης, ένα bit ελέγχου READ/WRITE και ένα bit επιβεβαίωσης (ACK). Εάν το bit READ/WRITE είναι λογικό 1 πρέπει να εκτελεστεί μια λειτουργία ανάγνωσης, διαφορετικά θα πρέπει να εκτελεστεί μια λειτουργία εγγραφής. Το MSBit του byte διεύθυνσης μεταδίδεται πρώτο. Ένα πακέτο διευθύνσεων που αποτελείται από μια διεύθυνση slave και ένα bit READ ή WRITE ονομάζεται SLA+R ή SLA+W αντίστοιχα. Όταν ένας Slave αναγνωρίσει ότι έχει διευθυνσιοδοτηθεί, τότε πρέπει να στείλει σήμα αναγνώρισης (ACK) κρατώντας το SDA χαμηλά στον ένατο κύκλο ρολογιού (κύκλος ACK). Εάν ο Slave είναι απασχολημένος ή για κάποιο άλλο λόγο δεν μπορεί να εξυπηρετήσει το αίτημα του Master, η γραμμή SDA θα πρέπει να παραμείνει ψηλά στον κύκλο ρολογιού ACK. Το Master μπορεί στη συνέχεια να μεταδώσει μια συνθήκη STOP ή μια συνθήκη REPEATED START για να ξεκινήσει μια νέα μετάδοση. Οι διευθύνσεις των συσκευών του διαύλου μπορούν να ειχωρηθούν ελεύθερα, αλλά η διεύθυνση '0000 000' προορίζεται για μια γενική κλήση. Όταν εκδίδεται μια γενική κλήση, όλοι οι Slave θα πρέπει να ανταποκρίνονται τραβώντας τη γραμμή SDA χαμηλά στον κύκλο ACK. Μια γενική κλήση χρησιμοποιείται όταν ένας Master επιθυμεί να μεταδώσει το ίδιο μήνυμα σε πολλούς Slave στο σύστημα. Όταν η διεύθυνση γενικής κλήσης ακολουθούμενη από ένα bit εγγραφής μεταδίδεται στο δίαυλο, όλοι οι Slave που έχουν ρυθμιστεί για να επιβεβαιώνουν τη γενική κλήση θα τραβήξουν τη γραμμή SDA χαμηλά στον κύκλο ACK. Τα ακόλουθα πακέτα δεδομένων θα ληφθούν από όλους τους Slave που αναγνώρισαν τη γενική κλήση. Όλες οι διευθύνσεις της μορφής «1111xxxx» θα πρέπει να δεσμευτούν για μελλοντικούς σκοπούς.

Πακέτα δεδομένων

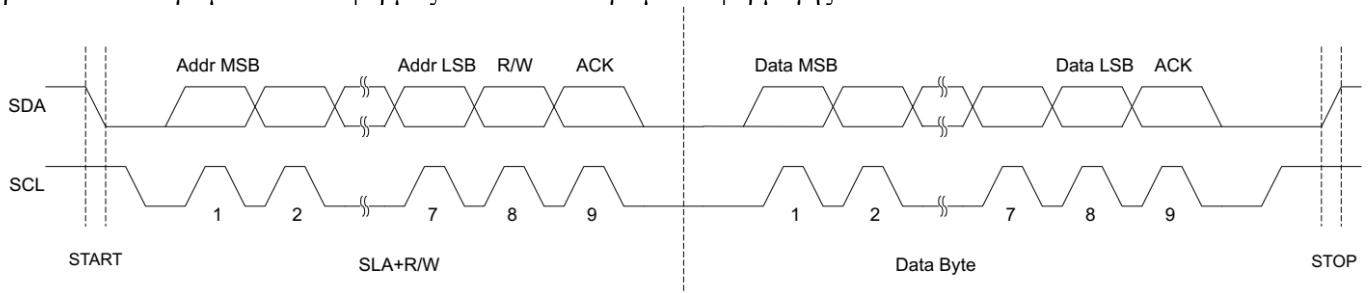
Όλα τα πακέτα δεδομένων που μεταδίδονται στο δίαυλο TWI έχουν μήκος εννέα bit, 8 bit δεδομένων και ένα bit επιβεβαίωσης. Το MSBit του byte δεδομένων μεταδίδεται πρώτο. Κατά τη μεταφορά δεδομένων, ο Master δημιουργεί το ρόλοι, το START και το STOP, ενώ ο δέκτης είναι υπεύθυνος για την επιβεβαίωση της παραλαβής (ACK). Εάν ο δέκτης παραλείψει το ACK τότε αυτό ισοδυναμεί με μη επιβεβαίωση (Not Acknowledge, NACK). Όταν ο δέκτης λάβει το τελευταίο byte ή για κάποιο λόγο δεν μπορεί να λάβει άλλα byte, θα πρέπει να ενημερώσει τον πομπό στέλνοντας ένα NACK μετά το τελευταίο byte.



Σχήμα 5.5 Μορφή πακέτου δεδομένων

Τυπική μετάδοση δεδομένων

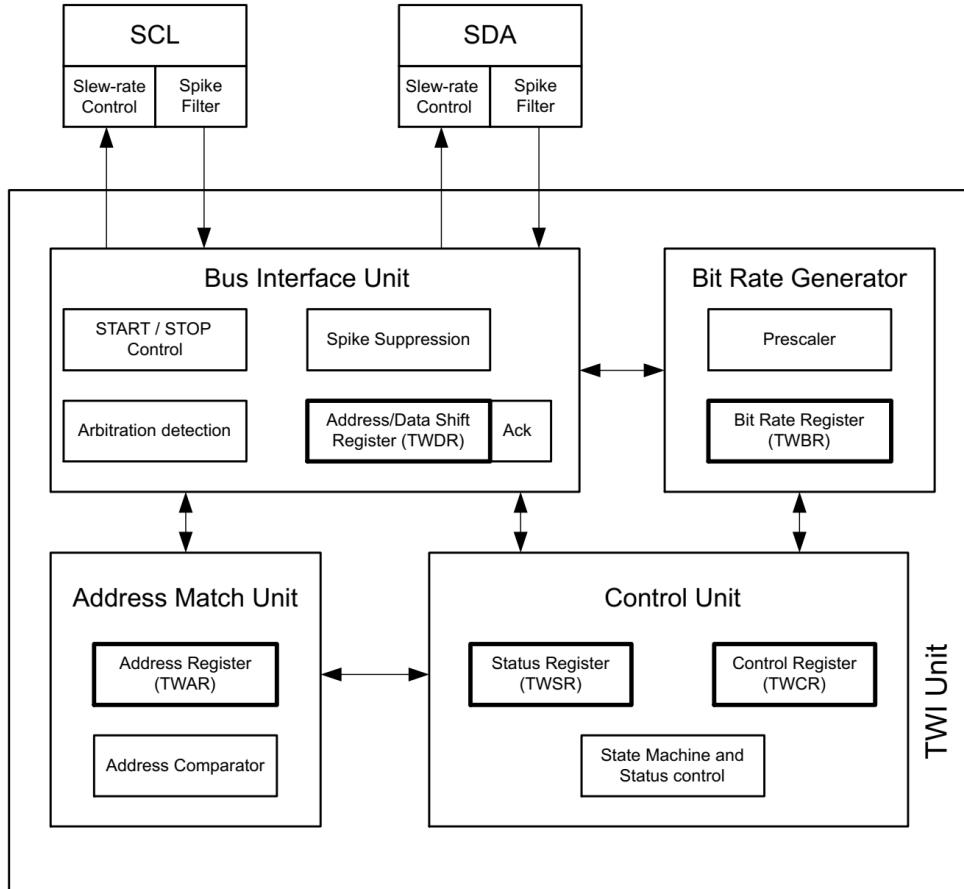
Μια μετάδοση αποτελείται από μια συνθήκη START, ένα πακέτο διεύθυνσης (Slave address(SLA) +R/W), ένα ή περισσότερα πακέτα δεδομένων και μια συνθήκη STOP. Ένα κενό μήνυμα, που αποτελείται από ένα START ακολουθούμενο από ένα STOP, είναι μη έγκυρο. Ο Slave μπορεί να παρατείνει τη χαμηλή περίοδο του ρολογιού κρατώντας τη γραμμή SCL χαμηλά. Αυτό είναι χρήσιμο εάν η ταχύτητα ρολογιού που έχει ρυθμιστεί από τον Master είναι πολύ γρήγορη για το Slave ή εάν το Slave χρειάζεται επιπλέον χρόνο για επεξεργασία μεταξύ των μεταδόσεων δεδομένων. Το παρακάτω σχήμα απεικονίζει μια τυπική μετάδοση δεδομένων. Σημειώστε ότι πολλά byte δεδομένων μπορούν να μεταδοθούν μεταξύ του SLA+R/W και της συνθήκης STOP, ανάλογα με το πρωτόκολλο λογισμικού που εφαρμόζεται από το λογισμικό εφαρμογής.



Σχήμα 5.6 Τυπική μετάδοση δεδομένων

Διεπαφή TWI στον ATmega328PB

Η Διεπαφή TWI στον ATmega328PB αποτελείται από πολλά υποσυστήματα, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 5.7 Διάγραμμα διεπαφής TWI στον ATmega328PB

Στη συνέχεια παρουσιάζονται οι καταχωρητές του ATmega328PB που σχετίζονται με τη λειτουργία του TWI.

Register	Offset	Reset Value	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
TWBR0	0xB8	0b00000000	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn	TWBRn		
TWSR0	0xB9	0b1111X00	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS[1:0]			
TWAR0	0xBA	0b00000010	TWA[6:0]									
TWDRO	0xBB	0b00000001	TWD[7:0]									
TWCRO	0xBC	0b000000X0	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE		
TWAMR0	0xBD	0b0000000X	TWAM[6:0]									

Πίνακας 5.1 Οι καταχωρητές του ATmega328PB που σχετίζονται με τη λειτουργία του TWI

TWBR0 (Offset 0xB8), TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0
	TWBRn							
Access	R/W							
Reset	0	0	0	0	0	0	0	0

Σχήμα 5.8 TWBR0, TWI Bit Rate Register

TWBR0[7:0] Ρυθμίζουν τη συχνότητα του σήματος SCL σύμφωνα με την ακόλουθη εξίσωση:

$$\text{Συχνότητα SCL} = \frac{\text{Συχνότητα ρολογιού CPU}}{16 + 2 \cdot (\text{TWBR0}) \cdot \text{PrescalerValue}}$$

Η τιμή του Prescaler καθορίζεται από τα bits TWPS[1:0] του καταχωρητή TWSR0.

TWSR0(Offset 0xB9), TWI Status Register

Bit	7	6	5	4	3	2	1	0
	TWS7	TWS6	TWS5	TWS4	TWS3		TWPS[1:0]	
Access	R	R	R	R	R		R/W	R/W
Reset	1	1	1	1	1		0	0

Σχήμα 5.9 TWSR0, TWI Status Register

TWPS[1:0] Η τιμή του Prescaler για τη ρύθμιση της συχνότητας του σήματος SCL καθορίζεται από τα bits TWPS[1:0] σύμφωνα με τον παρακάτω πίνακα:

TWPS[1:0]	Prescaler Value
00	1
01	4
10	16
11	64

Πίνακας 5.2 Bit rate Prescaler Values

TWS[7:3] αντικατοπτρίζουν την κατάσταση λειτουργίας του σειριακού διαύλου TWI για καθένα από τους τέσσερις κύριους τρόπους λειτουργίας:

1. Status Codes for Master Transmitter Mode

TWSR	Status of the Bus and Interface Hardware
0x08	A START has been transmitted
0x10	A repeated START has been transmitted
0x18	SLA+W has been transmitted; ACK has been received
0x20	SLA+W has been transmitted; NOT ACK has been received
0x28	Data byte has been transmitted; ACK has been received
0x30	Data byte has been transmitted; NOT ACK has been received
0x38	Arbitration lost in SLA+W or data bytes

2. Status codes for Master Receiver Mode

TWSR	Status of the Bus and Interface Hardware
0x08	A START has been transmitted
0x10	A repeated START has been transmitted
0x38	Arbitration lost in SLA+R or NOT ACK bit
0x40	SLA+R has been transmitted; ACK has been received
0x48	SLA+R has been transmitted; NOT ACK has been received
0x50	Data byte has been received; ACK has been returned
0x58	Data byte has been received; NOT ACK has been returned

3. Status Codes for Slave Transmitter Mode

TWSR	Status of the Bus and Interface Hardware
0xA8	Own SLA+R has been received; ACK has been returned
0xB0	Arbitration lost in SLA+R/W; Own SLA+R has been received; ACK has been returned;
0xB8	Data byte has been transmitted; ACK has been received;
0xC0	Data byte has been transmitted; NOT ACK has been received;
0xC8	Last data byte has been transmitted; ACK has been received;

4. Status codes for Slave Receiver Mode

TWSR	Status of the Bus and Interface Hardware
0x60	Own SLA+W has been received; ACK has been returned;
0x68	Arbitration lost in SLA+R/W; Own SLA+W has been received; ACK has been returned;
0x70	General call address has been received; ACK has been returned;
0x78	Arbitration lost in SLA+R/W; General call address has been received; ACK has been returned;
0x80	Data has been received; ACK has been returned;
0x88	Data has been received; NOT ACK has been returned;
0x90	General call data has been received; ACK has been returned;
0x98	General call data has data has been received; NOT ACK has been returned;
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave;

TWAR0 (Offset 0xBA), TWI (Slave) Address Register

Bit	7	6	5	4	3	2	1	0
TWA[6:0]								
Access	R/W							
Reset	0	0	0	0	0	0	1	0

Σχήμα 5.10 TWAR0, TWI (Slave) Address

TWA[6:0]: είναι η 7-bit διεύθυνση του slave

TWGCE: Χρησιμοποιείται για να επιτρέψει την αναγνώριση της γενικής διεύθυνσης κλήσης (0x00).

TWDR0 (Offset 0xBB), TWI Data Register

Bit	7	6	5	4	3	2	1	0
TWD[7:0]								
Access	R/W							
Reset	0	0	0	0	0	0	0	1

Σχήμα 5.11 TWDR0, TWI Data Register

TWD[7:0] Byte δεδομένων που θα μεταδοθεί ή το τελευταίο byte δεδομένων που ελήφθη.

TWCR0 (Offset 0xBC), TWI Control Register

Bit	7	6	5	4	3	2	1	0
Access	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE
Reset	R/W	R/W	R/W	R/W	R	R/W		R/W

Σχήμα 5.12 TWCR0, TWI Control Register

TWINT: Σημαία διακοπής TWI. Τίθεται όταν το TWI ολοκληρώσει την τρέχουσα εργασία του και αναμένει την απόκριση του λογισμικού εφαρμογής. Αν TWINT=1 τότε η χαμηλή περίοδος του SCL επιμηκύνεται. Η σημαία TWINT μηδενίζεται μόνο από το λογισμικό γράφοντας λογικό 1 σε αυτήν. Αν το TWINT μηδενιστεί τότε ξεκινά τη λειτουργία του TWI. Όλες οι προσβάσεις στα TWAR0, TWSR0 και TWDR0 πρέπει να έχουν ολοκληρωθεί πριν από την εκκαθάριση αυτής της σημαίας.

TWEA: Αυτό το bit ελέγχει τη δημιουργία του παλμού επιβεβαίωσης. Εάν TWEA=1 το ACK δημιουργείται στον δίαυλο εάν έχει ληφθεί η ατομική διεύθυνση του slave ή έχει ληφθεί μια γενική κλήση και ταυτόχρονα TWAR0.TWGCE =1 ή έχει ληφθεί ένα byte δεδομένων σε λειτουργία Master Receiver ή Slave Receiver.

Γράφοντας το bit TWEA στο μηδέν, η συσκευή μπορεί ουσιαστικά να αποσυνδεθεί από το δίαυλο προσωρινά. Στη συνέχεια, η αναγνώριση διεύθυνσης μπορεί να ενεργοποιηθεί ξανά, θέτοντας ξανά το bit TWEA.

TWSTA: Συνθήκη START. Η εφαρμογή θέτει το TWSTA όταν επιθυμεί να γίνει Master. Το υλικό TWI ελέγχει εάν ο δίαυλος είναι διαθέσιμος και δημιουργεί μια συνθήκη START στο δίαυλο εάν είναι ελεύθερος. Ωστόσο, εάν ο δίαυλος δεν είναι ελεύθερος, το TWI περιμένει μέχρι να εντοπιστεί μια συνθήκη STOP και στη συνέχεια δημιουργεί μια νέα συνθήκη START για να δώσει το δικαίωμα του Master. Το TWSTA πρέπει να μηδενιστεί από το λογισμικό όταν έχει γίνει εκπομπή του START.

TWSTO: Συνθήκη STOP. Σε λειτουργία Master θα δημιουργήσει μια συνθήκη STOP στον δίαυλο. Όταν η συνθήκη STOP εκτελεστεί στο δίαυλο, το TWSTO μηδενίζεται αυτόματα. Στη λειτουργία Slave το TWSTO τίθεται για την ανάκτηση από μια συνθήκη σφάλματος. Αυτό δεν θα δημιουργήσει μια συνθήκη STOP, αλλά το TWI επιστρέφει σε μια προκαθορισμένη λειτουργία μη διευθυνσιοδοτούμενου Slave και απελευθερώνει τις γραμμές SCL και SDA.

TWWC: Σημαία σύγκρουσης εκπομπής στο δίαυλο. Το bit TWWC τίθεται όταν γίνεται προσπάθεια εγγραφής του TWDR0 ενώ TWCR0.TWINT =0. Αυτή η σημαία μηδενίζεται γράφοντας στον καταχωρητή TWDR0 ενώ TWINT=1.

TWEN: Ενεργοποίηση του TWI. Όταν TWEN=1 το TWI θα αναλαμβάνει τον έλεγχο των ακροδεκτών που είναι συνδεδεμένες με τα σήματα SCL και SDA. Επίσης ενεργοποιεί τους περιοριστές του ρυθμού μεταβολής της τάσης και τα φίλτρα θορύβου σε αυτούς τους ακροδέκτες. Εάν αυτό το TWEN μηδενιστεί το TWI απενεργοποιείται και όλες οι μεταδόσεις του τερματίζονται.

TWIE: Ενεργοποίηση της διακοπής του TWI. Όταν TWIE =1 και SREG.I=1 το αίτημα διακοπής του TWI θα ενεργοποιηθεί για όσο διάστημα η σημαία TWCR0.TWINT είναι υψηλή.

TWAMR0 (Offset 0xBD), TWI Slave Address Mask Register

Bit	7	6	5	4	3	2	1	0
TWAM[6:0]								
Access	R/W							
Reset	0	0	0	0	0	0	0	

Σχήμα 5.13 TWAMR0 , TWI Slave Address Mask Register

TWAM[6:0]: Το TWAMR0 μπορεί να φορτωθεί με μια μάσκα για τη διεύθυνση του Slave (7 bit). Κάθε ένα από τα bit TWAMR[6:0] μπορεί να κρύψει(απενεργοποιήσει) τα αντίστοιχα bit διεύθυνσης στον καταχωρητή διευθύνσεων TWAR0. Εάν το bit μάσκας έχει οριστεί σε λογικό 1, τότε η διάταξη αντιστοίχισης διεύθυνσης αγνοεί τη σύγκριση μεταξύ του εισερχόμενου bit διεύθυνσης και του αντίστοιχου bit στο TWAR0.

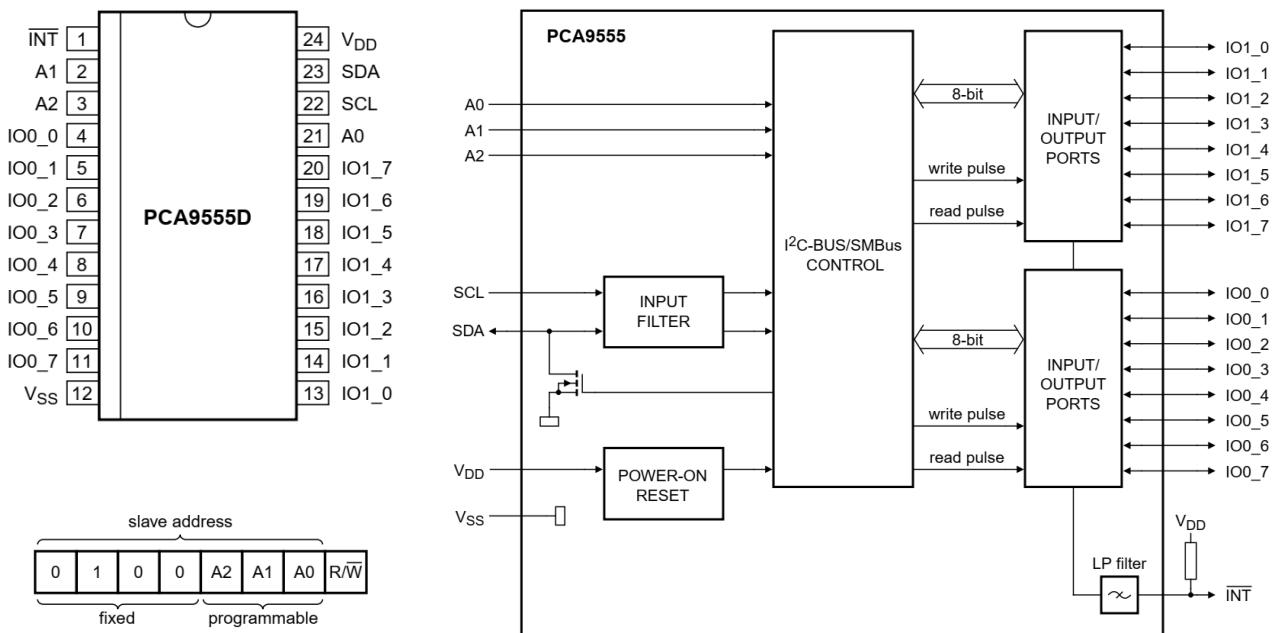
Παράδειγμα μετάδοσης ενός byte δεδομένων σε ένα slave.

Το TWI βασίζεται σε διακοπές. Όταν τεθεί το TWINT, τότε το TWI έχει ολοκληρώσει μια λειτουργία και ο καταχωρητής κατάστασης TWSR0 περιέχει μια τιμή που υποδεικνύει την τρέχουσα κατάσταση του TWI διαύλου. Το λογισμικό εφαρμογής ρυθμίζει τους καταχωρητές TWCR0 και TWDR0 για τον επόμενο κύκλο του διαύλου TWI .

	ASSEMBLY CODE	C CODE	ΣΧΟΛΙΑ
1	<code>Ldi r16,1<<TWINT) (1<<TWSTA) (1<<TWEN) out TWCR0, r16</code>	<code>TWCR0 = (1<<TWINT) (1<<TWSTA) (1<<TWEN)</code>	Αποστολή START
2	<code>Call wait1 Wait1: in r16, TWCR0 sbrs r16, TWINT rjmp wait1 ret</code>	<code>while (!(TWCR0 & (1<<TWINT)));</code>	Αναμονή έως TWINT=1 που σημαίνει ότι το START μεταδόθηκε.
3a	<code>in r16, TWSR0 andi r16, 0xF8 cpi r16, START brne ERROR</code>	<code>if ((TWSR0 & 0xF8) != START) ERROR();</code>	Αν TWSR0.TWS[7:3] δείχνει κωδικό διαφορετικό από START τότε υπάρχει σφάλμα.
3b	<code>ldi r16, SLA_W out TWDR0, r16 ldi r16, (1<<TWINT) (1<<TWEN) out TWCR0, r16</code>	<code>TWDR0 = SLA_W; TWCR0 = (1<<TWINT) (1<<TWEN);</code>	Φορτώστε το SLA_W στον καταχωρητή TWDR0 . Μηδένισε το TWCR.WINT για να ξεκινήσει η μετάδοση.
4	Call wait1	<code>while (!(TWCR0 & (1<<TWINT)));</code>	
5a	<code>in r16, TWSR0 andi r16, 0xF8 cpi r16, MT_SLA_ACK brne ERROR</code>	<code>if ((TWSR0 & 0xF8) != MT_SLA_ACK) ERROR();</code>	Αν TWSR0.TWS[7:3] δείχνει κωδικό διαφορετικό από SLA_ACK τότε υπάρχει σφάλμα.
5b	<code>ldi r16, DATA out TWDR0, r16 ldi r16, (1<<TWINT) (1<<TWEN) out TWCR, r16</code>	<code>TWDR0 = DATA; TWCR0 = (1<<TWINT) (1<<TWEN);</code>	Φορτώστε τα DATA στον καταχωρητή TWDR0 . Μηδένισε το TWCR.WINT για να ξεκινήσει η μετάδοση.
6	Call wait1	<code>while (!(TWCR0 & (1<<TWINT)));</code>	
7a	<code>in r16, TWSR0 andi r16, 0xF8 cpi r16, MT_DATA_ACK brne ERROR</code>	<code>if ((TWSR0 & 0xF8) != MT_DATA_ACK) ERROR();</code>	Αν TWSR0.TWS[7:3] δείχνει κωδικό διαφορετικό από DATA_ACK τότε υπάρχει σφάλμα.
7b	<code>ldi r16, (1<<TWINT) (1<<TWEN) (1<<TWSTO) out TWCR0, r16</code>	<code>TWCR0 = (1<<TWINT) (1<<TWEN) (1<<TWSTO);</code>	Αποστολή STOP

Ολοκληρωμένο PCA9555

Το PCA9555 είναι ένα ολοκληρωμένο που παρέχει δύο πόρτες εισόδου/εξόδου γενικής χρήσης, IO0 και IO1, των 8-bit η κάθε μια, για εφαρμογές I²C-bus. Διαθέτει μία έξοδο διακοπής (open-drain) που ενεργοποιείται όταν η κατάσταση εισόδου διαφέρει από την αντίστοιχη κατάσταση του καταχωρητή θύρας εισόδου και χρησιμοποιείται για να υποδείξει ότι μια κατάσταση εισόδου έχει μεταβληθεί. Κατά την εφαρμογή της τάσεις τροφοδοσίας γίνεται επαναφορά της συσκευής και οι καταχωρητές τίθενται στις προεπιλεγμένες τιμές τους. Η I²C διεύθυνση του PCA9555 είναι 7-bit εκ των οποίων τα 4bit είναι σταθερά και τα υπόλοιπα τρία ρυθμίζονται από τους ακροδέκτες A0, A1 και A2. Στο ntuAboard_G1 οι ακροδέκτες A0, A1 και A2 είναι συνδεδεμένοι στη γη (0 Volt) οπότε η διεύθυνση του ενσωματωμένου PCA9555 είναι 0b0100000. Το διάγραμμα, οι ακροδέκτες και η I²C διεύθυνση του PCA9555 φαίνονται στο παρακάτω σχήμα:



Σχήμα 5.14 Διάγραμμα, ακροδέκτες και I²C διεύθυνση του PCA9555

Καταχωρητές του PCA9555

Κατά τη διάρκεια μιας μετάδοσης εγγραφής, το πρώτο byte που ακολουθεί το byte της διεύθυνσης(command byte) χρησιμοποιείται ως δείκτης για να προσδιορίσει ποιος από τους παρακάτω καταχωρητές θα γραφεί ή θα διαβαστεί σύμφωνα με τον παρακάτω πίνακα:

Command	Register
0	Input port 0
1	Input port 1
2	Output port 0
3	Output port 1
4	Polarity Inversion port 0
5	Polarity Inversion port 1
6	Configuration port 0
7	Configuration port 1

Πίνακας 5.3 Command byte

Οι καταχωρητές Input port 0 και Input port 1 είναι θύρες μόνο εισόδου. Αντικατοπτρίζουν τα εισερχόμενα λογικά επίπεδα των ακροδεκτών, ανεξάρτητα από το εάν ένας ακροδέκτης ορίζεται ως είσοδος ή έξοδος. Οι εγγραφές σε αυτούς τους καταχωρητές δεν έχουν κανένα αποτέλεσμα. Η προεπιλεγμένη τιμή «X» καθορίζεται από το εξωτερικά εφαρμοζόμενο λογικό επίπεδο.

Input port 0 Register

Bit	7	6	5	4	3	2	1	0
Symbol	I0.7	I0.6	I0.5	I0.4	I0.3	I0.2	I0.1	I0.0
Default	X	X	X	X	X	X	X	X

Input port 1 Register

Bit	7	6	5	4	3	2	1	0
Symbol	I1.7	I1.6	I1.5	I1.4	I1.3	I1.2	I1.1	I1.0
Default	X	X	X	X	X	X	X	X

Οι καταχωρητές Output port 0 και Output port 1 είναι θύρες μόνο εξόδου. Αντικατοπτρίζουν τα εξερχόμενα λογικά επίπεδα των ακροδεκτών, που ορίζονται ως έξοδοι. Οι αναγνώσεις από αυτούς τους καταχωρητές αντικατοπτρίζουν την τιμή που βρίσκεται στα flip-flop που ελέγχουν την επιλογή εξόδου, όχι την πραγματική τιμή των ακροδεκτών.

Output port 0 register

Bit	7	6	5	4	3	2	1	0
Symbol	O0.7	O0.6	O0.5	O0.4	O0.3	O0.2	O0.1	O0.0
Default	1	1	1	1	1	1	1	1

Output port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	O1.7	O1.6	O1.5	O1.4	O1.3	O1.2	O1.1	O1.0
Default	1	1	1	1	1	1	1	1

Οι καταχωρητές Polarity Inversion port 0 και Polarity Inversion port 1 επιτρέπουν την αντιστροφή της πολικότητας των δεδομένων των καταχωρητών της θύρας εισόδου. Εάν έχει τεθεί κάποιο bit σε έναν από αυτούς τους καταχωρητές, η πολικότητα δεδομένων της θύρας εισόδου αντιστρέφεται.

Polarity Inversion port 0 register

Bit	7	6	5	4	3	2	1	0
Symbol	N0.7	N0.6	N0.5	N0.4	N0.3	N0.2	N0.1	N0.0
Default	0	0	0	0	0	0	0	0

Polarity Inversion port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	N1.7	N1.6	N1.5	N1.4	N1.3	N1.2	N1.1	N1.0
Default	0	0	0	0	0	0	0	0

Οι καταχωρητές Configuration port 0 και Configuration port 1 διαμορφώνουν τις κατευθύνσεις των ακροδεκτών I/O. Εάν έχει τεθεί ένα bit σε έναν από αυτούς τους καταχωρητές, ο αντίστοιχος ακροδέκτης της θύρας ενεργοποιείται

ως είσοδος. Υπάρχει μια αντίσταση υψηλής τιμής(pull-up resistor) για πρόσδεσης κάθε ακροδέκτη στο VDD. Κατά την επαναφορά, οι θύρες της συσκευής είναι είσοδοι με αντιστάσεις πρόσδεσης στο VDD.

Configuration port 0 register

Bit	7	6	5	4	3	2	1	0
Symbol	C0.7	C0.6	C0.5	C0.4	C0.3	C0.2	C0.1	C0.0
Default	1	1	1	1	1	1	1	1

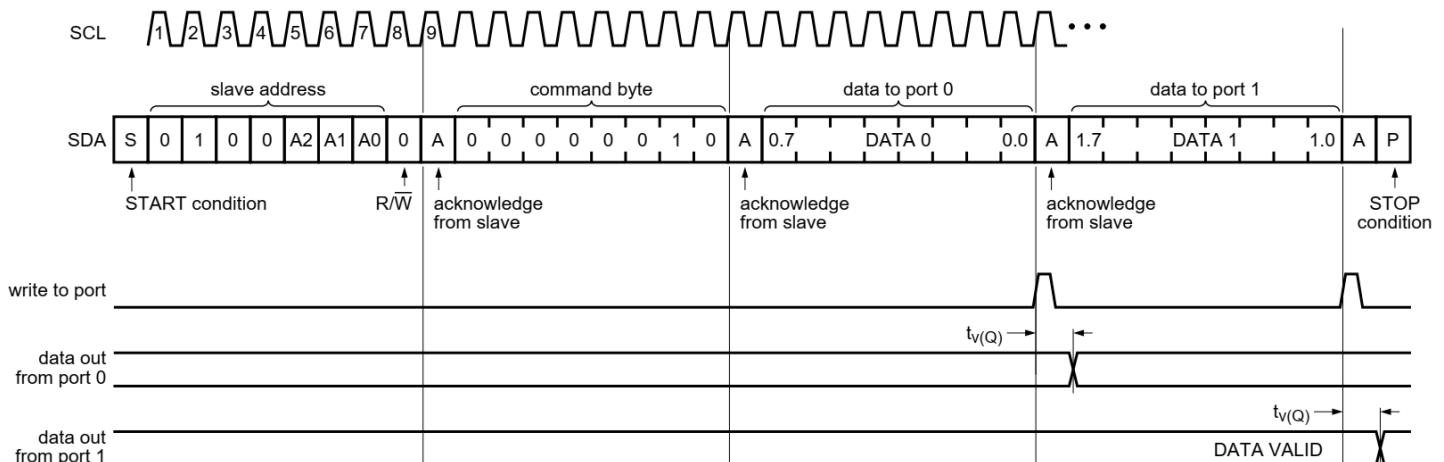
Configuration port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	C1.7	C1.6	C1.5	C1.4	C1.3	C1.2	C1.1	C1.0
Default	1	1	1	1	1	1	1	1

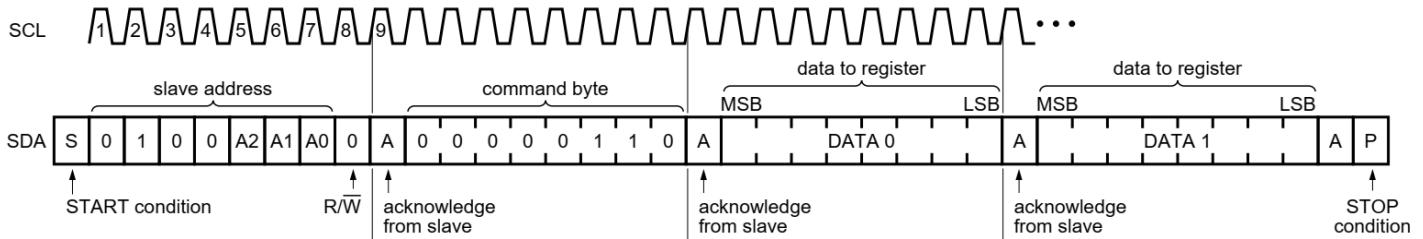
Τα δεδομένα μεταδίδονται στο PCA9555 στέλνοντας τη διεύθυνση της συσκευής και μηδενίζοντάς το λιγότερο σημαντικό bit. Το byte εντολής αποστέλλεται μετά τη διεύθυνση και καθορίζει ποιος καταχωρητής θα λάβει τα δεδομένα που ακολουθούν. Οι οκτώ καταχωρητές τουPCA9555 έχουν ρυθμιστεί να λειτουργούν ως τέσσερα ζεύγη καταχωρητών. Τα τέσσερα ζεύγη είναι θύρες εισόδου, θύρες εξόδου, θύρες αναστροφής πολικότητας και θύρες διαμόρφωσης.

Εγγραφή στους καταχωρητές θυρών

Μετά την αποστολή δεδομένων σε έναν καταχωρητή, το επόμενο byte δεδομένων θα σταλεί στον άλλο καταχωρητή του ζεύγουν. Για παράδειγμα, εάν το πρώτο byte αποσταλεί στην Output Port 1 (register 3) τότε το επόμενο byte θα αποθηκευτεί στη Output Port 0 (register 2). Δεν υπάρχει περιορισμός στον αριθμό των byte δεδομένων που αποστέλλονται σε μία μετάδοση εγγραφής. Με αυτόν τον τρόπο, κάθε καταχωρητής 8-bit μπορεί να ενημερώνεται ανεξάρτητα από τους άλλους καταχωρητές. Στα επόμενα δύο σχήματα απεικονίζεται η διαδικασία εγγραφής στα Output port registers και στα Configuration registers.



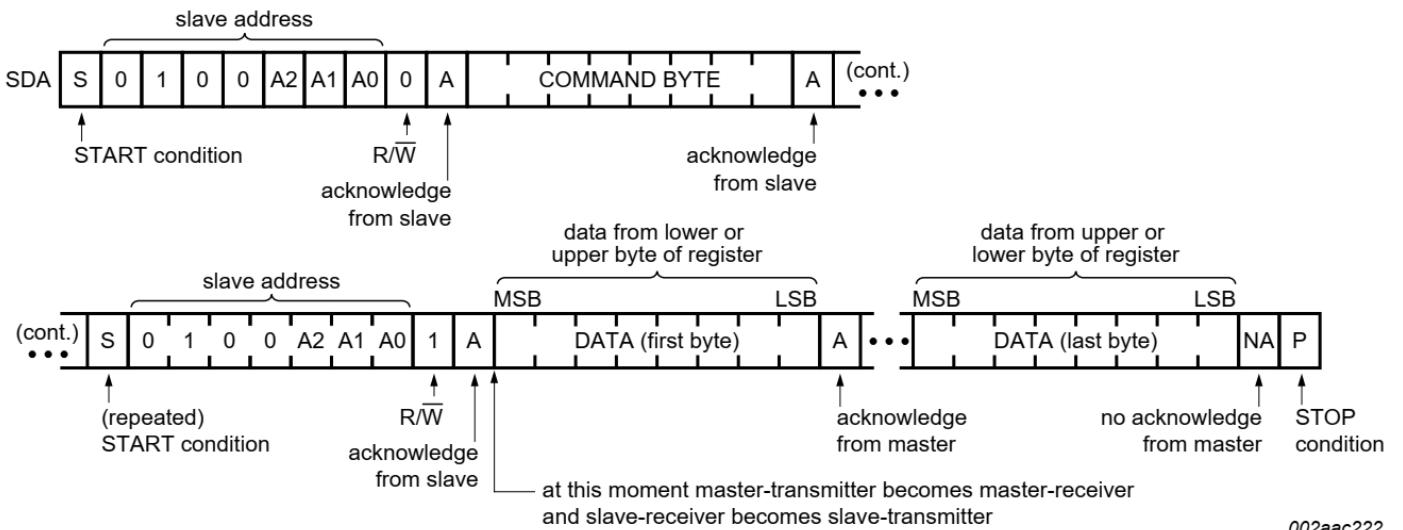
Σχήμα 5.15 Write to Output port registers



Σχήμα 5.16 Write to Configuration registers

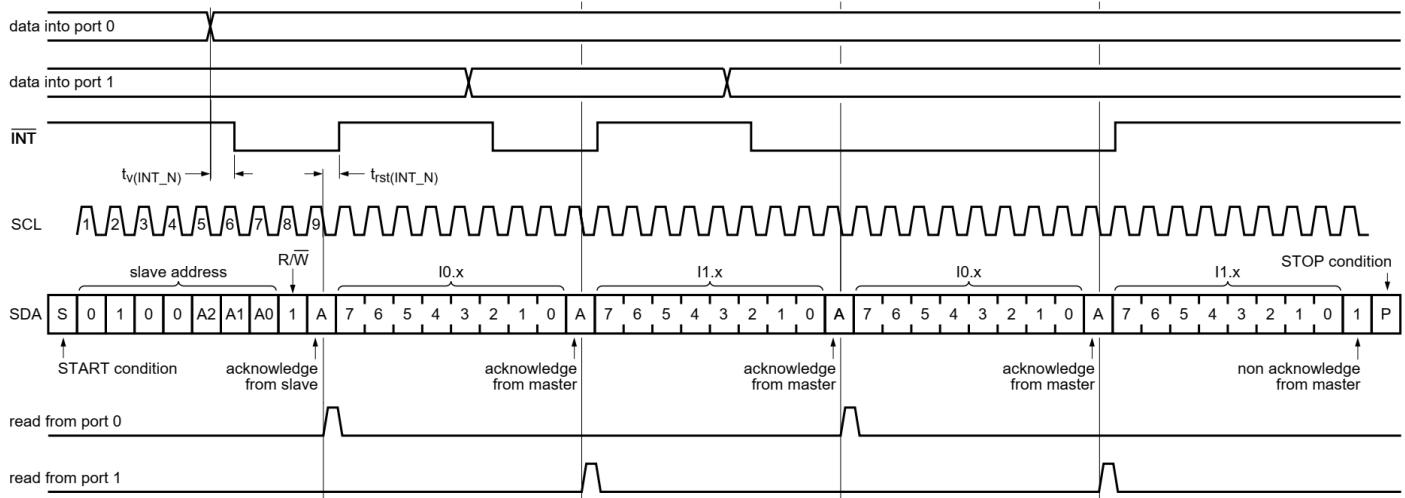
Ανάγνωση των καταχωρητών των θυρών

Για να διαβαστούν τα δεδομένα από το PCA9555, ο master του I2C διαύλου πρέπει πρώτα να στείλει τη διεύθυνση του PCA9555 με το λιγότερο σημαντικό bit να έχει οριστεί σε λογικό 0. Μετά τη διεύθυνση αποστέλλεται το command byte και καθορίζει σε ποιον καταχωρητή θα γίνει προσπέλαση. Μετά από επανεκκίνηση, η διεύθυνση της συσκευής αποστέλλεται ξανά, αλλά αυτή τη φορά το λιγότερο σημαντικό bit ορίζεται σε ένα λογικό 1. Τα δεδομένα από τον καταχωρητή που ορίζεται από command byte θα σταλούν στη συνέχεια από το PCA9555. Μετά την ανάγνωση του πρώτου byte, μπορούν να διαβαστούν επιπλέον byte, αλλά τα δεδομένα θα αντικατοπτρίζουν τώρα τις πληροφορίες στον άλλο καταχωρητή του ζεύγους. Δεν υπάρχει περιορισμός στον αριθμό των byte δεδομένων που λαμβάνονται σε μία μετάδοση ανάγνωσης, αλλά στο τελικό byte που λαμβάνεται, ο master του διαύλου δεν πρέπει να στείλει ACK. Στα επόμενα σχήματα απεικονίζεται η διαδικασία ανάγνωσης από τους διάφορους καταχωρητές.



Σχήμα 5.17 Read from register

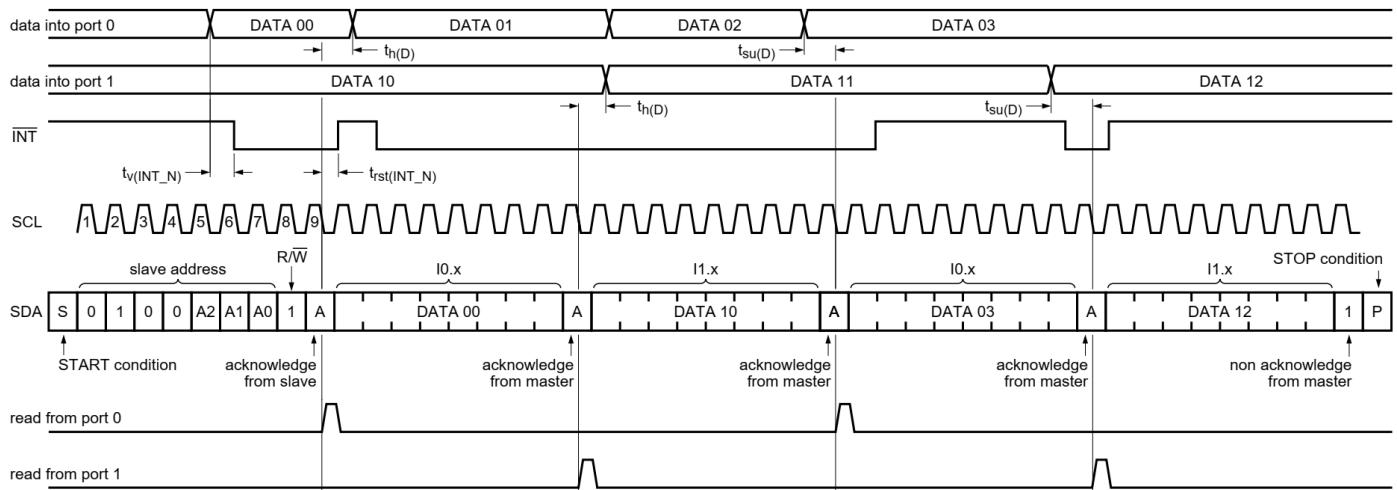
Η μεταφορά μπορεί να διακοπεί ανά πάσα στιγμή με αποστολή STOP.



Σχήμα 5.18 Read Input port register, scenario 1

Η μεταφορά δεδομένων μπορεί να διακοπεί ανά πάσα στιγμή με μια συνθήκη STOP. Όταν συμβεί αυτό, τα δεδομένα που υπάρχουν στην τελευταία φάση επιβεβαίωσης είναι έγκυρα.

Υποτίθεται ότι το command byte έχει προηγουμένως ρυθμιστεί σε «00» (Διάβασμα Input Port register).



Σχήμα 5.19 Read Input port register, scenario 2

Η μεταφορά δεδομένων μπορεί να διακοπεί ανά πάσα στιγμή με μια συνθήκη STOP. Όταν συμβεί αυτό, τα δεδομένα που υπάρχουν στην τελευταία φάση επιβεβαίωσης είναι έγκυρα.

Υποτίθεται ότι το command byte έχει προηγουμένως ρυθμιστεί σε «00» (Διάβασμα Input Port register).

Παράδειγμα 5.1

Στο παρακάτω παράδειγμα ρυθμίζεται η θύρα επέκτασης 0 του ολοκληρωμένου PCA9555 έτσι ώστε η λογική ή κατάσταση των ακροδεκτών να εναλλάσσεται μεταξύ λογικού 0 και λογικού 1. Ο χρόνος παραμονής σε καθεμία από τις δύο αυτές λογικές καταστάσεις είναι ένα δευτερόλεπτο. Το ολοκληρωμένο PCA9555 επικοινωνεί με τον μικροελεγκτή ATmega328PB μέσω της σειριακής Θύρας TWI.

```
#define F_CPU 16000000UL

#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>

#define PCA9555_0_ADDRESS 0x40      //A0=A1=A2=0 by hardware
#define TWI_READ    1              // reading from twi device
#define TWI_WRITE   0              // writing to twi device
#define SCL_CLOCK  100000L         // twi clock in Hz

//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2

// PCA9555 REGISTERS
typedef enum {
    REG_INPUT_0          = 0,
    REG_INPUT_1          = 1,
    REG_OUTPUT_0         = 2,
    REG_OUTPUT_1         = 3,
    REG_POLARITY_INV_0  = 4,
    REG_POLARITY_INV_1  = 5,
    REG_CONFIGURATION_0 = 6,
    REG_CONFIGURATION_1 = 7
} PCA9555_REGISTERS;

//----- Master Transmitter/Receiver -----
#define TW_START          0x08
#define TW_REP_START       0x10

//----- Master Transmitter -----
#define TW_MT_SLA_ACK     0x18
#define TW_MT_SLA_NACK    0x20
#define TW_MT_DATA_ACK    0x28

//----- Master Receiver -----
#define TW_MR_SLA_ACK     0x40
#define TW_MR_SLA_NACK    0x48
#define TW_MR_DATA_NACK   0x58
```

```

#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)

//initialize TWI clock
void twi_init(void)
{
    TWSR0 = 0;           // PRESCALER_VALUE=1
    TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}

// Read one byte from the twi device (request more data from device)
unsigned char twi_readAck(void)
{
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

//Read one byte from the twi device, read is followed by a stop condition
unsigned char twi_readNak(void)
{
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address)
{
    uint8_t twi_status;

    // send START condition
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // wait until transmission completed
    while(!(TWCR0 & (1<<TWINT)));

    // check value of TWI Status Register.
    twi_status = TW_STATUS & 0xF8;
    if( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;

    // send device address
    TWDR0 = address;
    TWCR0 = (1<<TWINT) | (1<<TWEN);

    // wait until transmission completed and ACK/NACK has been received
    while(!(TWCR0 & (1<<TWINT)));
}

```

```

// check value of TWI Status Register.
twi_status = TW_STATUS & 0xF8;
if( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
{
    return 1;
}

return 0;
}

// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
    uint8_t twi_status;

    while ( 1 )
    {
        // send START condition
        TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR0 & (1<<TWINT)));

        // check value of TWI Status Register.
        twi_status = TW_STATUS & 0xF8;
        if( (twi_status != TW_START) && (twi_status != TW REP START)) continue;

        // send device address
        TWDR0 = address;
        TWCR0 = (1<<TWINT) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR0 & (1<<TWINT)));

        // check value of TWI Status Register.
        twi_status = TW_STATUS & 0xF8;
        if( (twi_status == TW_MT_SLA_NACK )||(twi_status ==TW_MR_DATA_NACK) )
        {
            /* device busy, send stop condition to terminate write operation */
            TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

            // wait until stop condition is executed and bus released
            while(TWCR0 & (1<<TWSTO));

            continue;
        }
        break;
    }
}

```

```

// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write( unsigned char data )
{
    // send data to the previously addressed device
    TWDR0 = data;
    TWCR0 = (1<<TWINT) | (1<<TWEN);

    // wait until transmission completed
    while(!(TWCR0 & (1<<TWINT)));
    if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
    return 0;
}

// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
//          1 failed to access device
unsigned char twi_rep_start(unsigned char address)
{
    return twi_start( address );
}

// Terminates the data transfer and releases the twi bus
void twi_stop(void)
{
    // send stop condition
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

    // wait until stop condition is executed and bus released
    while(TWCR0 & (1<<TWSTO));
}

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_write(value);
    twi_stop();
}

uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
{
    uint8_t ret_val;

    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
    ret_val = twi_readNak();
    twi_stop();
}

```

```

    return ret_val;
}

int main(void) {
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output

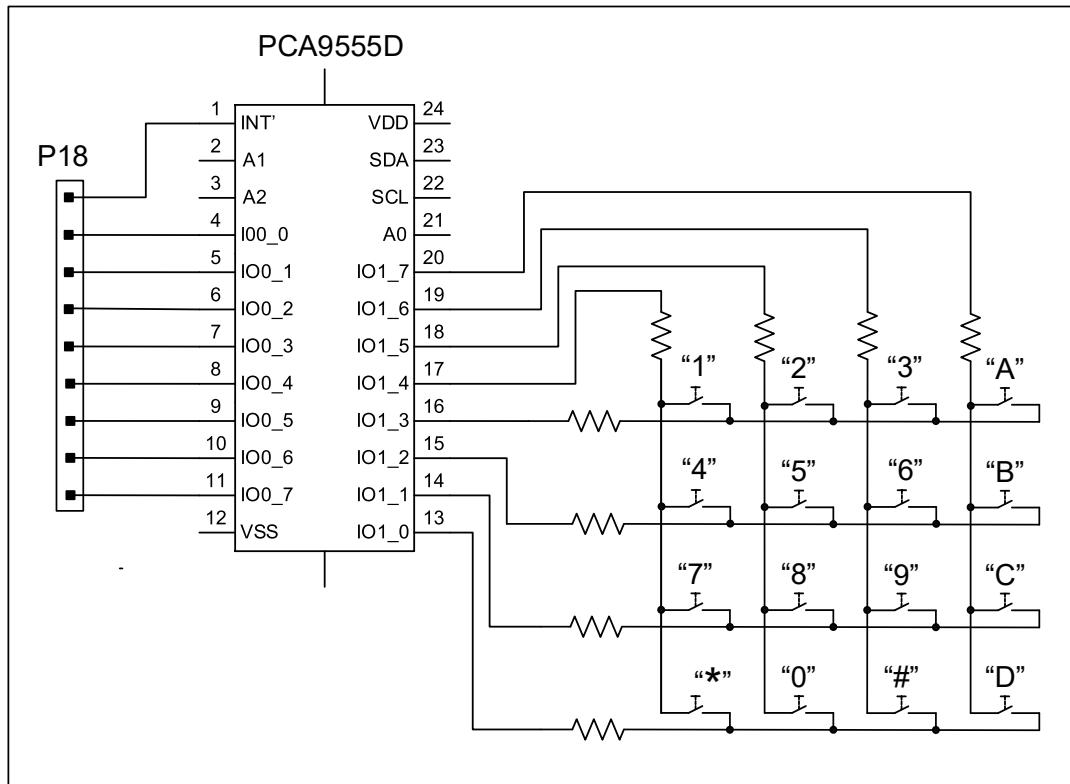
    while(1)
    {
        PCA9555_0_write(REG_OUTPUT_0, 0x00);
        _delay_ms(1000);

        PCA9555_0_write(REG_OUTPUT_0, 0xFF);
        _delay_ms(1000);
    }
}

```

Διασύνδεση του ολοκληρωμένου PCA9555D

Στο παρακάτω σχήμα εμφανίζεται η διασύνδεση του ολοκληρωμένου PCA9555D στο ntuAboard_G1.



Τα ζητούμενα της 5^{ης} εργαστηριακής άσκησης

Ζήτημα 5.1

Προκειμένου να μπορέσει να υπάρξει οπτική απεικόνιση της λογικής κατάστασης των ακροδεκτών IO0_0 και IO0_1, της θύρας επέκτασης 0, να ρυθμιστούν ως έξοδοι και να συνδεθούν, μέσω καλωδίων, με τους ακροδέκτες LED_PD0 και LED_PD1 του κονέκτορα J18 αντίστοιχα.

Να υλοποιηθούν για το μικροελεγκτή ATmega328PB, σε γλώσσα C, οι λογικές συναρτήσεις:

$$F0 = (AB' + CBD)'$$

$$F1 = (A+C) \cdot (B \cdot D)$$

Οι μεταβλητές εισόδου δίνονται στα bit **PORTB [3:0]** ($A = PORTB.0$, $B = PORTB.1$, $C = PORTB.2$, $D = PORTB.3$) του ntuAboard_G1.

Οι τιμές των $F0$ - $F1$ να εμφανίζονται αντίστοιχα στους ακροδέκτες IO0_0 και IO0_1 του ολοκληρωμένου επέκτασης θυρών PCA9555, στο ntuAboard_G1.

Ζήτημα 5.2

Προκειμένου να μπορέσει να υπάρξει οπτική απεικόνιση της λογικής κατάστασης των ακροδεκτών IO0_0 έως IO0_3, της θύρας επέκτασης 0, να ρυθμιστούν ως έξοδοι και να συνδεθούν, μέσω καλωδίων, με τους ακροδέκτες LED_PD0 έως LED_PD3 του κονέκτορα J18 αντίστοιχα.

Ο ακροδέκτης IO1_3 της θύρας επέκτασης 1 να ρυθμιστεί ως έξοδος ενώ οι ακροδέκτες IO1_4 έως IO0_7 να ρυθμιστούν ως είσοδοι.

Να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος:

- όταν πιέζεται το πλήκτρο “1” να ανάβει το led PD0
- όταν πιέζεται το πλήκτρο “2” να ανάβει το led PD1
- όταν πιέζεται το πλήκτρο “3” να ανάβει το led PD2
- όταν πιέζεται το πλήκτρο “A” να ανάβει το led PD3

Αν δεν πιέζεται κανένα πλήκτρο τότε όλα τα led να παραμένουν σβηστά.

Θεωρείστε δεδομένο ότι κάθε φορά πιέζεται μόνο ένα πλήκτρο.

Ζήτημα 5.3

Να συνδεθεί η LCD οθόνη 2x16 χαρακτήρων, δια μέσω του κονέκτορα J19, στην θύρα επέκτασης 0 του ολοκληρωμένου PCA9555 και να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος θα απεικονίζει στην οθόνη το όνομα και το επίθετο σας.



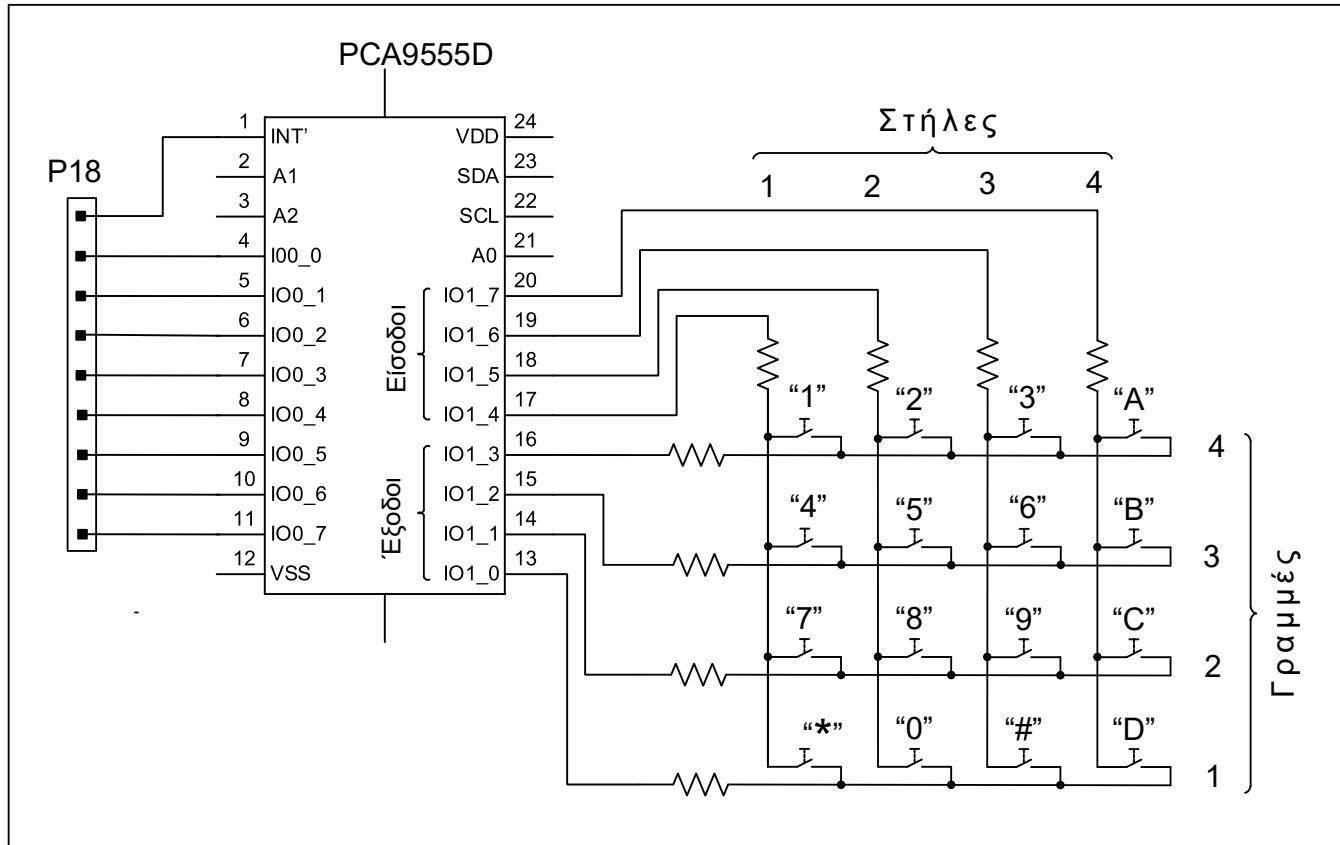
ΑΘΗΝΑ, 14/11/2025

6^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικρούπολογιστών"
Χρήση πληκτρολογίου 4x4 σε θύρα επέκτασης στον AVR

Εξέταση – Επίδειξη: Παρασκευή 21/11/2025
Προθεσμία για παράδοση Έκθεσης: Τρίτη 25/11/2025 (23:59)

Πληκτρολόγιο 4x4

Το πληκτρολόγιο 4x4 της κάρτας ntuAboard έχει τέσσερις γραμμές και τέσσερις στήλες δηλαδή συνολικά 16 πλήκτρα. Όταν πατηθεί κάποιο πλήκτρο ενώνονται η γραμμή και στήλη που αντιστοιχούν σε αυτό το πλήκτρο. Η ακριβής διάταξη του πληκτρολογίου φαίνεται στο παρακάτω σχήμα:



Σχήμα 6.1 Σύνδεση του πληκτρολογίου 4x4 στην εξωτερική θύρα IO1 του PCA9555

Το πληκτρολόγιο είναι συνδεδεμένο στην εξωτερική θύρα εισόδου/εξόδου γενικής χρήσης IO1 του PCA9555, το οποίο επικοινωνεί με τον μικροελεγκτή ATmega328PB δια μέσω της διεπαφής TWI0 και η διεύθυνση του είναι 0b0100000. Οι δυο γραμμές του διαύλου TWI0 συνδέονται στους ακροδέκτες PC4 (SDA) και PC5 (SCL) του ATmega328PB. Οι αντιστάσεις πρόσδεσης (pull-up resistors) συνδέονται στο δίαυλο με την τοποθέτηση βραχυκλωτήρων στους κονέκτορες J12 και J13. Όταν ο δίαυλος TWI είναι ενεργός τότε οι ακροδέκτες PC4(SDA) και PC5(SCL) δεν μπορούν να χρησιμοποιηθούν για άλλο σκοπό. Όταν ένας ακροδέκτης, οποιασδήποτε από τις δύο θύρες του PCA9555, ρυθμιστεί ως είσοδος τότε ο ακροδέκτης αυτός προσδένεται στην τάση VDD μέσω μιας αντίστασης υψηλής τιμής(pull-up resistor).

Ανάγνωση του πληκτρολογίου

Μία μέθοδος ανάγνωσης του πληκτρολογίου είναι η εξής:

Διαδοχικά κάθε μία από τις γραμμές του πληκτρολογίου, οι οποίες έχουν ρυθμιστεί ως έξοδοι και είναι συνδεδεμένες με τους ακροδέκτες IO1[3:0] της θύρας IO1 του PCA9555, τίθεται σε λογικό 0 (χαμηλή στάθμη τάσης) ενώ ταυτόχρονα οι υπόλοιπες γραμμές έχουν τεθεί σε λογικό 1. Για κάθε μία από τις γραμμές, που έχει τεθεί σε λογικό 0, εκτελείται μία μικρή χρονοκαθυστέρηση και μετά διαβάζονται οι ακροδέκτες IO1[7:4] οι οποίοι αντιστοιχούν στις στήλες του πληκτρολογίου και έχουν ρυθμιστεί ως είσοδοι. Αν δεν υπάρχει πιεσμένος διακόπτης, λόγω των pull-up αντιστάσεων, οι ακροδέκτες εισόδου βρίσκονται σε κατάσταση λογικού 1. Τα πλήκτρα που είναι πατημένα κάθε φορά μπορούν να εντοπιστούν γνωρίζοντας πια γραμμή έχει τεθεί σε λογικό 0 και διαβάζοντας ταυτόχρονα ποιες από τις στήλες βρίσκονται σε λογικό 0.

Όταν ο χρήστης πατάει ένα πλήκτρο, αυτό μπορεί να μείνει πιεσμένο για μεγάλο χρονικό διάστημα. Για το λόγο αυτό απαιτείται ιδιαίτερος χειρισμός για να εντοπιστούν τα πλήκτρα που έχουν πατηθεί ως ολοκληρωμένη ενέργεια και όχι απλά αυτά που είναι κάθε φορά πατημένα. Αυτό επιτυγχάνεται εάν κληθεί μία ρουτίνα που διαβάζει την κατάσταση των πλήκτρων και αποθηκεύει το αποτέλεσμα στη μνήμη του μικροελεγκτή. Στη συνέχεια καλείτε ξανά η ίδια ρουτίνα, για νέο έλεγχο της κατάστασης των πλήκτρων. Μια σύγκριση των δύο καταστάσεων του πληκτρολογίου θα αποκαλύψει τις διαφορές στην κατάσταση των διακοπών. Το χρονικό διάστημα ανάμεσα στις διαδοχικές κλήσεις της συνάρτησης διαβάσματος του πληκτρολογίου είναι κρίσιμο, διότι καθορίζει το χρόνο που θα πρέπει να μείνει πιεσμένος ένας διακόπτης 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. Οι τυπικές τιμές καθυστέρησης είναι της τάξης των 10 έως 20 msec.

Ο παρακάτω κώδικας μπορεί να χρησιμοποιηθεί για να γίνει εγγραφή ή ανάγνωση ενός από τους καταχωρητές ελέγχου του ολοκληρωμένου PCA9555:

```
#define F_CPU 16000000UL

#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>

#define PCA9555_0_ADDRESS 0x40          //A0=A1=A2=0 by hardware
#define TWI_READ    1                  // reading from twi device
#define TWI_WRITE   0                  // writing to twi device
#define SCL_CLOCK  100000L             // twi clock in Hz

//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2

// PCA9555 REGISTERS
typedef enum {
    REG_INPUT_0      = 0,
    REG_INPUT_1      = 1,
    REG_OUTPUT_0     = 2,
    REG_OUTPUT_1     = 3,
    REG_POLARITY_INV_0 = 4,
    REG_POLARITY_INV_1 = 5,
    REG_CONFIGURATION_0 = 6,
    REG_CONFIGURATION_1 = 7
} PCA9555_REGISTERS;

//----- Master Transmitter/Receiver -----
#define TW_START        0x08
#define TW_REP_START    0x10

//----- Master Transmitter -----
#define TW_MT_SLA_ACK   0x18
#define TW_MT_SLA_NACK  0x20
#define TW_MT_DATA_ACK  0x28

//----- Master Receiver -----
#define TW_MR_SLA_ACK   0x40
#define TW_MR_SLA_NACK  0x48
#define TW_MR_DATA_NACK 0x58

#define TW_STATUS_MASK  0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)

//initialize TWI clock
```

```

void twi_init(void)
{
    TWSR0 = 0;           // PRESCALER_VALUE=1
    TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}

// Read one byte from the twi device (request more data from device)
unsigned char twi_readAck(void)
{
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

//Read one byte from the twi device, read is followed by a stop condition
unsigned char twi_readNak(void)
{
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address)
{
    uint8_t twi_status;

    // send START condition
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // wait until transmission completed
    while(!(TWCR0 & (1<<TWINT)));

    // check value of TWI Status Register.
    twi_status = TW_STATUS & 0xF8;
    if( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;

    // send device address
    TWDR0 = address;
    TWCR0 = (1<<TWINT) | (1<<TWEN);

    // wail until transmission completed and ACK/NACK has been received
    while(!(TWCR0 & (1<<TWINT)));

    // check value of TWI Status Register.
    twi_status = TW_STATUS & 0xF8;
    if( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
    {
        return 1;
    }
}

```

```

        return 0;
    }

// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
    uint8_t  twi_status;

    while ( 1 )
    {
        // send START condition
        TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR0 & (1<<TWINT)));

        // check value of TWI Status Register.
        twi_status = TW_STATUS & 0xF8;
        if ( (twi_status != TW_START) && (twi_status != TW REP START)) continue;

        // send device address
        TWDR0 = address;
        TWCR0 = (1<<TWINT) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR0 & (1<<TWINT)));

        // check value of TWI Status Register.
        twi_status = TW_STATUS & 0xF8;
        if ( (twi_status == TW_MT_SLA_NACK )||(twi_status ==TW_MR_DATA_NACK) )
        {
            /* device busy, send stop condition to terminate write operation */
            TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

            // wait until stop condition is executed and bus released
            while(TWCR0 & (1<<TWSTO));

            continue;
        }
        break;
    }
}

// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write( unsigned char data )
{
    // send data to the previously addressed device
    TWDR0 = data;
}

```

```

TWCR0 = (1<<TWINT) | (1<<TWEN);

// wait until transmission completed
while(!(TWCR0 & (1<<TWINT)));
if (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
return 0;
}

// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
//          1 failed to access device
unsigned char twi_rep_start(unsigned char address)
{
    return twi_start( address );
}

// Terminates the data transfer and releases the twi bus
void twi_stop(void)
{
    // send stop condition
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

    // wait until stop condition is executed and bus released
    while(TWCR0 & (1<<TWSTO));
}

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_write(value);
    twi_stop();
}

uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
{
    uint8_t ret_val;

    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
    ret_val = twi_readNak();
    twi_stop();

    return ret_val;
}

```

Τα ζητούμενα της 6^{ης} εργαστηριακής άσκησης

Ζήτημα 6.1

Να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος να διαβάζει το πληκτρολόγιο της ntuAboard_G1, και να έχει τη δομή που περιγράφεται ακολούθως:

- Να δημιουργηθεί μια συνάρτηση με όνομα scan_row η οποία να ελέγχει μια γραμμή του πληκτρολογίου για πιεσμένους διακόπτες.
- Να δημιουργηθεί μια συνάρτηση με όνομα scan_keypad η οποία θα καλεί διαδοχικά την scan_row 4 φορές και θα ελέγχει ολόκληρο το πληκτρολόγιο.
- Να δημιουργηθεί μια συνάρτηση με όνομα scan_keypad_rising_edge η οποία θα καλείτε συνεχόμενα και θα κρατάει σε μια μεταβλητή με όνομα pressed_keys μία καταγραφή των πατημένων πλήκτρων.

Η συνάρτηση scan_keypad_rising_edge θα καλεί τη συνάρτηση scan_keypad και θα αποθηκεύει τα πλήκτρα που είναι πατημένα σε μια 16μπιτη μεταβλητή με όνομα pressed_keys_tempo. Προκειμένου να αντιμετωπιστεί το πρόβλημα του σπινθηρισμού, που παρουσιάζουν οι διακόπτες, θα καλεί ξανά την scan_keypad, μετά από χρονικό διάστημα της τάξης των 10 έως 20 mSec και θα συγκρίνει την τρέχουσα εικόνα των πατημένων πλήκτρων με αυτή που είναι αποθηκευμένη στη μεταβλητή pressed_keys_tempo και όσα πλήκτρα διαφέρουν θα απορρίπτονται τροποποιώντας ανάλογα τη μεταβλητή pressed_keys_tempo.

Η τροποποιημένη τιμή της μεταβλητής pressed_keys_tempo θα συγκρίνεται με την καταγραφή που είναι αποθηκευμένη στη μεταβλητή pressed_keys από προηγούμενη κλίση έτσι ώστε να εντοπίσει ποια πλήκτρα έχουν πατηθεί μόλις τώρα, απορρίπτοντας τα πλήκτρα που ήταν πατημένα και στην προηγούμενη κλήση της scan_keypad_rising_edge. Η τωρινή καταγραφή των πατημένων πλήκτρων στη μεταβλητή pressed_keys_tempo θα αντικαθιστά την καταγραφή στην μεταβλητή pressed_keys για χρήση στην επόμενη κλήση.

- Να δημιουργηθεί μια συνάρτηση με όνομα 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. Θεωρείστε δεδομένο ότι κάθε φορά μπορεί να είναι πατημένο μόνο ένα πλήκτρο.
- Να αντιστοιχήσετε τέσσερα από τα πλήκτρα του πληκτρολογίου σε τέσσερα led σύμφωνα με τον παρακάτω πίνακα:

“4”	“2”	“3”	“B”
Led στο PB1	Led στο PB2	Led στο PB3	Led στο PB4

Κάθε φορά που θα πιέζεται κάποιο από τα παραπάνω τέσσερα πλήκτρα να ανάβει το αντίστοιχο led. To led θα παραμένει αναμμένο όσο διάστημα το πλήκτρο είναι πατημένο.

Θεωρείστε δεδομένο ότι κάθε φορά μπορεί να είναι πατημένο μόνο ένα πλήκτρο.

Ζήτημα 6.2

Να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος θα κάνει χρήση των συναρτήσεων του ζητήματος 6.1 και θα απεικονίζει στην οθόνη LCD 2x16 το χαρακτήρα που αντιστοιχεί στο πλήκτρο που πατήθηκε τελευταίο.

Θεωρείστε δεδομένο ότι κάθε φορά μπορεί να πατηθεί μόνο ένα πλήκτρο.

Ζήτημα 6.3

Γράψτε ένα πρόγραμμα «ηλεκτρονικής κλειδαριάς» το οποίο να ανάβει τα 6 leds PB0 έως PB5 για 3 sec, όταν δοθεί από το keypad 4×4 ο διψήφιος αριθμός της ομάδας σας (π.χ. 09). Εάν κάποιος σπουδαστής δεν έχει αριθμό ομάδας να κάνει χρήση του αριθμού 99. Αν δεν έχουν δοθεί σωστά τα δύο ψηφία του αριθμού της ομάδας τότε τα leds PB0 έως PB5 να αναβοσβήνουν(500 mSec ανάμενα, 500 mSec σβηστά) για 6 sec. Ανεξάρτητα από το χρονικό διάστημα για το οποίο θα μείνει πατημένο ένα πλήκτρο, το πρόγραμμά θα πρέπει να θεωρεί ότι πατήθηκε μόνο μια φορά. Μετά το πάτημα δύο αριθμών το πρόγραμμα να μην δέχεται για 5 sec άλλον αριθμό. Το πρόγραμμα να είναι συνεχόμενης λειτουργίας. Δώστε το διάγραμμα ροής και το πρόγραμμα σε γλώσσα C.



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

AΘΗΝΑ, 21/11/2025

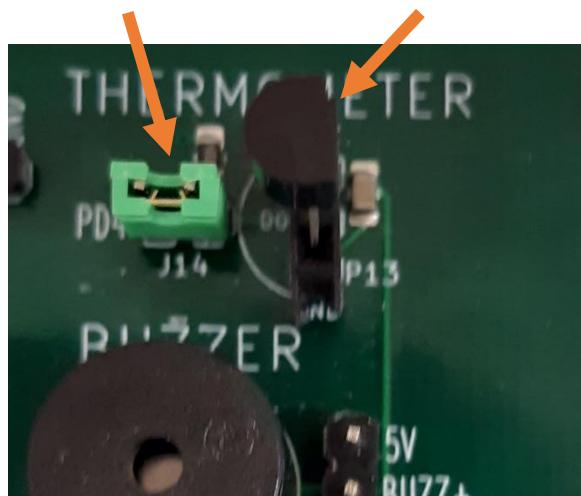
7^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"
Αισθητήρας Θερμοκρασίας DS1820 στην κάρτα ntuAboard_G1

Εξέταση – Επίδειξη: Παρασκευή 28/11/2025
Προθεσμία για παράδοση Έκθεσης: Τρίτη 2/12/2025 (23:59)

Χρήση Αισθητήρα Θερμοκρασίας DS1820 και Σειριακή Επικοινωνία 1-wire

Η σειριακή επικοινωνία ενός καλωδίου (1-wire serial communication) είναι ένα πρωτόκολλο που επιτρέπει τη μεταφορά δεδομένων από μία μόνο καλωδιακή σύνδεση. Η διαδικασία συντονίζεται από ένα μικροελεγκτή σε ρόλο master. Το πλεονέκτημα του συγκεκριμένου πρωτοκόλλου είναι ότι απαιτεί έναν μόνο ακροδέκτη του μικροελεγκτή. Όλες οι συσκευές που συνδέονται σε αυτόν τον ακροδέκτη έχουν έναν προκαθορισμένο κωδικό slave, που επιτρέπει στο μικροελεγκτή να αναγνωρίζει εύκολα με ποια συσκευή επικοινωνεί κάθε φορά.

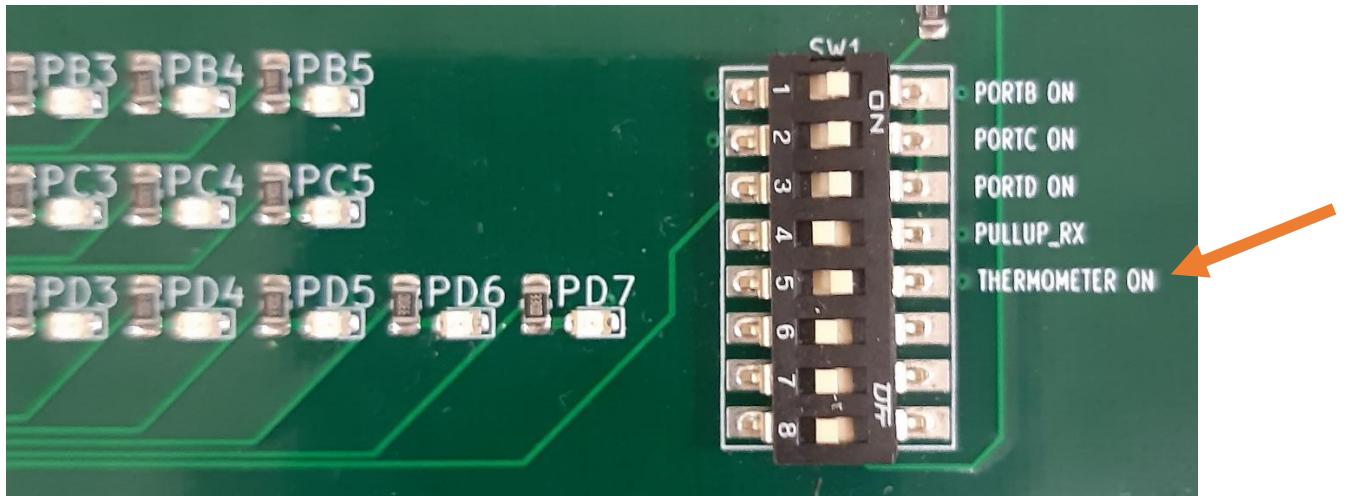
Η αναπτυξιακή πλακέτα ntuAboard_G1 είναι εφοδιασμένη με τον αισθητήρα θερμοκρασίας DS1820 (σχήμα 7.1) που χρησιμοποιεί το πρωτόκολλο σειριακής επικοινωνίας ενός καλωδίου. Μετράει θερμοκρασίες από -55°C έως 125°C και παρέχει ακρίβεια ±0.5°C στην περιοχή θερμοκρασιών από -10°C έως 85°C. Για την τροφοδοσία του απαιτείται συνεχής τάση από 3V έως 5V. Ο αισθητήρας χρειάζεται μέγιστο χρόνο 750msec για να υπολογίσει τη θερμοκρασία με ακρίβεια 9 bit.



Σχήμα 7.1 Αισθητήρας θερμοκρασίας DS1820.

Στην αναπτυξιακή πλακέτα *ntuAboard_G1* ο αισθητήρας θερμοκρασίας DS1820 τοποθετείται στη δεξιά πάνω γωνία και για να συνδεθεί με τον ακροδέκτη PD4 του μικροελεγκτή, απαιτείται η χρήση ενός βραχυκυκλωτήρα στο κονέκτορα J14, όπως φαίνεται στο σχήμα 7.1.

Η σύνδεση της τροφοδοσίας στον αισθητήρα θερμοκρασίας DS1820 γίνεται δια μέσω του DIP Switch SW1-5 (THERMOMETER ON) όπως φαίνεται στο Σχήμα 7.2 παρακάτω



Σχήμα 7.2 Σύνδεση τροφοδοσίας στον αισθητήρα DS1820

Η επικοινωνία του μικροελεγκτή με τον αισθητήρα θερμοκρασίας πραγματοποιείται δια μέσω του ακροδέκτη PD4. Η τοποθέτηση του αισθητήρα στην πλακέτα *ntuAboard_G1* γίνεται σε ειδική βάση, προσέχοντας το τυπωμένο ημικύκλιο στην πλακέτα να συμπίπτει με την ημικυκλική πλευρά του αισθητήρα, όπως στο σχήμα 7.1. Το ολοκληρωμένο DS1820 δεν είναι ένας απλός αισθητήρας αλλά μια σύνθετη συσκευή με ενσωματωμένο μετατροπέα ADC και πρωτόκολλο επικοινωνίας. Αν συνδεθεί λανθασμένα θα πάθει βλάβη.

Η επικοινωνία ενός καλωδίου είναι ένα πολύ απλό πρωτόκολλο που διαβάζει και γράφει σειριακά δεδομένα τηρώντας συγκεκριμένα χρονικά παράθυρα, όπως φαίνεται στο τεχνικό εγχειρίδιο του DS1820. Η σωστή λειτουργία του πρωτοκόλλου απαιτεί και την ύπαρξη μιας αντίστασης πρόσδεσης (pull-up) στον ακροδέκτη PD4.

Στη συνέχεια περιγράφεται ο κώδικας για την επικοινωνία του DS1820 με το μικροελεγκτή ATmega328PB που βρίσκεται πάνω στην αναπτυξιακή πλακέτα *ntuAboard_G1*.

Η ρουτίνα αρχικοποίησης **one_wire_reset** εκπέμπει έναν παλμό αρχικοποίησης στη γραμμή και ανιχνεύει εάν υπάρχει κάποια συνδεδεμένη συσκευή. Ο ακροδέκτης PD4 αρχικά ρυθμίζεται ως έξοδος και τίθεται σε κατάσταση λογικού 0, για 480usec. Στη συνέχεια μετά από 100usec, διαβάζεται η κατάσταση της θύρας PD, και ακολουθεί μία επιπλέον χρονική καθυστέρηση 380usec ώστε να ολοκληρωθεί ο κύκλος αρχικοποίησης. Αν κάποια συνδεδεμένη συσκευή αποκριθεί και θέσει το PD4 σε κατάσταση λογικού 0 τότε επιστρέφεται η τιμή 1 στον καταχωρητή r24, διαφορετικά επιστρέφεται η τιμή 0.

one_wire_reset:

```
sbi DDRD, PD4      ; set PD4 as output

cbi PORTD, PD4      ;
ldi r24, low(480)    ; 480 usec reset pulse
ldi r25, high(480)   ;
rcall wait_usec     ;

cbi DDRD, PD4      ; set PD4 as input
cbi PORTD, PD4      ; disable pull-up

ldi r24 ,100        ; wait 100 usec for connected devices
ldi r25 ,0           ; to transmit the presence pulse
rcall wait_usec     ;

in r24, PIND         ; read PORTD

push r24             ; save PORTD

ldi r24, low(380)    ;
ldi r25, high(380)   ; wait for 380 usec
rcall wait_usec     ;

pop r25              ; retrieve PORTD

clr r24              ; if a connected device is
sbrs r25, PD4        ; detected(PD4=0) return 1
ldi r24 ,0x01         ; else return 0
ret
```

Η ρουτίνα **one_wire_receive_bit** έχει ως αντικείμενο το διάβασμα ενός bit, το οποίο αποθηκεύεται στον καταχωρητή **r24**, από τον αισθητήρα DS1820 δια μέσω του ακροδέκτη PD4, σύμφωνα με το πρωτόκολλο επικοινωνίας 1-wire serial communication.. Ξεκινά με εντολές που ρυθμίζουν τη θύρα PD ως έξοδο και ο ακροδέκτης PD4 τίθεται σε λογικό 0 για 2usec. Στη συνέχεια ρυθμίζει τη θύρα PD ως είσοδο, περιμένει 10usec, διαβάζει τον ακροδέκτη PD4 και γράφει την τιμή που διάβασε στον καταχωρητή r24. Τέλος δημιουργείται μια ακόμη χρονική καθυστέρηση 49usec (48usec για ολοκλήρωση του κύκλου ανάγνωσης και 1usec για αποκατάσταση) και η ρουτίνα επιστέφει.

one_wire_receive_bit:

```
sbi DDRD, PD4      ; set PD4 as output

cbi PORTD, PD4      ;
ldi r24, 0x02        ; time slot 2 usec
ldi r25, 0x00        ;
rcall wait_usec      ;

cbi DDRD, PD4      ; set PD4 as input
cbi PORTD, PD4      ; disable pull-up

ldi r24 ,10          ;
ldi r25 ,0           ; wait 10 usec
rcall wait_usec      ;

clr r24              ;
sbic PIND, PD4      ; r24 = PD4
ldi r24, 1            ;
push r24

ldi r24, 49          ;
ldi r25, 0           ; delay 49 usec to meet the standards
rcall wait_usec      ;

pop r24
ret
```

Αντίστοιχα η ρουτίνα **one_wire_transmit_bit**, έχει ως αντικείμενο την εγγραφή ενός bit, το οποίο βρίσκεται καταχωρητή **r24**, στον αισθητήρα DS1820 δια μέσω του ακροδέκτη PD4. Ξεκινά με εντολές που ρυθμίζουν τη θύρα PORTD ως έξοδο και ο ακροδέκτης PD4 τίθεται σε λογικό 0 για 2usec. Στη συνέχεια ο ακροδέκτης PD4 διατηρείται σε λογικό 0 ή μεταβαίνει σε λογικό 1 ανάλογα με την τιμή του καταχωρητή r24 και περιμένει 58usec μέχρι ο αισθητήρας θερμοκρασίας να διαβάσει αυτή την τιμή. Τέλος, η θύρα PORTD ρυθμίζεται ως είσοδος, δημιουργείται μια ακόμη χρονική καθυστέρηση 1usec για αποκατάσταση και η ρουτίνα επιστέφει.

one_wire_transmit_bit:

```

push r24           ; save output bit

sbi DDRD, PD4    ; set PD4 as output

cbi PORTD, PD4   ;
ldi r24, 0x02     ; time slot 2 usec
ldi r25, 0x00     ;
rcall wait_usec   ;

pop r24           ; retrieve output bit

sbrc r24, 0        ;
sbi PORTD, PD4    ; PD4 = r24[0]
sbrs r24, 0        ;
cbi PORTD, PD4    ;

ldi r24, 58         ; wait 58 usec for connected
ldi r25, 0          ; device to sample the line
rcall wait_usec    ;

cbi DDRD, PD4    ; set PD4 as input
cbi PORTD, PD4    ; disable pull-up

ldi r24 ,0x01      ;
ldi r25 ,0x00      ; recovery time 1 usec
rcall wait_usec    ;

ret

```

Η ποντίνα **one_wire_receive_byte** έχει ως αντικείμενο τη λήψη ενός byte από τον αισθητήρα DS1820, καλώντας τη ρουτίνα `one_wire_receive_bit` 8 φορές ξεκινώντας από το λιγότερο σημαντικό bit. Το λαμβανόμενο byte αποθηκεύεται στον καταχωρητή **r24**.

`one_wire_receive_byte:`

```
ldi r27 ,8           ; 8 repetitions
clr r26

loop_:
rcall one_wire_receive_bit    ; r24[0] holds the received bit
lsr r26

sbrc r24, 0          ; logical or with 0x00 if returned bit is 0,
ldi r24 ,0x80        ; or
or r26, r24          ; with 0x80 if returned bit is 1

dec r27
brne loop_
mov r24, r26
ret
```

Η ποντίνα **one_wire_transmit_byte** έχει ως αντικείμενο την εγγραφή ενός byte στον αισθητήρα DS1820, καλώντας τη `one_wire_transmit_bit` 8 φορές ξεκινώντας από το λιγότερο σημαντικό bit. Το εκπεμπόμενο byte αποθηκεύεται στον καταχωρητή **r24**.

`one_wire_transmit_byte:`

```
mov r26, r24
ldi r27, 8           ; 8 repetitions

_one_more_:
clr r24
sbrc r26 ,0          ; load r24[0] with the transmitted bit
ldi r24 ,0x01        ;

rcall one_wire_transmit_bit

lsr r26
dec r27
brne _one_more_
ret
```

Τα ζητούμενα της 7^{ης} εργαστηριακής άσκησης

Ζήτημα 7.1 (α) Να υλοποιηθεί ένα πρόγραμμα σε γλώσσα C, για την επικοινωνία του μικροελεγκτή ATmega328PB με τον αισθητήρα θερμοκρασίας DS1820 στην κάρτα NtuAboard_G1.

Το πρόγραμμα να περιλαμβάνει μόνο τις στοιχειώδεις λειτουργίες, λαμβάνοντας υπόψη ότι στον ακροδέκτη PD4 είναι συνδεδεμένος μόνο ένας αισθητήρας. Οι στοιχειώδεις αυτές λειτουργίες στηρίζονται στις εντολές που δέχεται ο αισθητήρας και παρουσιάζονται στο τεχνικό εγχειρίδιον του DS1820.

Οι εντολές αυτές μπορούν να χωριστούν σε **εντολές μνήμης**, με τις οποίες επιλέγεται η συσκευή με την οποία θα γίνει η επικοινωνία, και **εντολές λειτουργίας**, με τις οποίες υλοποιούνται λειτουργίες που έχουν σχέση με τη μέτρηση θερμοκρασίας. Ενδιαφέροντας, ο αισθητήρας βρίσκεται σε κατάσταση **χαμηλής κατανάλωσης ισχύος** και επανέρχεται σε κατάσταση λειτουργίας με μια **εντολή αρχικοποίησης**.

Ειδικότερα να γραφεί ρουτίνα που να επιστρέφει στον διπλό καταχωρητή r25:r24 την τιμή της θερμοκρασίας και σε περίπτωση που δεν υπάρχει συνδεδεμένη συσκευή να επιστρέφει την τιμή 0x8000.

Ποιο αναλυτικά η ρουτίνα να περιλαμβάνει τα εξής:

- Αρχικοποίηση και έλεγχος αν υπάρχει συνδεδεμένη συσκευή με την εκτέλεση της ρουτίνας `one_wire_reset`.
- Χρήση της ρουτίνας `one_wire_transmit_byte` για αποστολή της εντολής **0xCC**, η οποία παρακάμπτει την επιλογή συσκευής από διάδρομο πολλών συσκευών, μιας και είναι δεδομένο ότι στην κάρτα NtuAboard_G1 υπάρχει μόνο μία συνδεδεμένη συσκευή.
- Χρήση της ρουτίνας `one_wire_transmit_byte` για αποστολή της εντολής **0x44**, η οποία ξεκινάει μια μέτρηση θερμοκρασίας. Όταν η μέτρηση θερμοκρασίας ολοκληρωθεί, το DS1820 σηματοδοτεί τον τερματισμό της μετατροπής με την αποστολή ενός bit με τιμή 1, το οποίο ελέγχεται με την εκτέλεση της ρουτίνας `one_wire_receive_bit`.
- Νέα αρχικοποίηση της συνδεόμενης συσκευής με την εκτέλεση της ρουτίνας `one_wire_reset`.
- Χρήση της ρουτίνας `one_wire_transmit_byte` για αποστολή της εντολής **0xCC**.
- Χρήση της ρουτίνας `one_wire_transmit_byte` για αποστολή της εντολής **0xBE** για ανάγνωση των 16 bit της μετρημένης θερμοκρασίας, η οποία πραγματοποιείται με κλίση της ρουτίνας `one_wire_receive_byte` δύο φορές και αποθήκευση της τιμής στο ζεύγος καταχωρητών r25:r24. Η τιμή που διαβάζεται, σύμφωνα με όσα αναφέρονται στο τεχνικό εγχειρίδιο, είναι ένας αριθμός σε μορφή συμπληρώματος ως προς 2. Τα 8 λιγότερα σημαντικά bit αντιστοιχούν σε θερμοκρασία βαθμών Κελσίου, με **ακρίβεια 0.5°C** ανά bit, και τα 8 περισσότερο σημαντικά αποτελούν επέκταση προσήμου (στην προκαθορισμένη λειτουργία του αισθητήρα).
- Για παράδειγμα, ο αριθμός 0x0032 αντιστοιχεί σε θερμοκρασία 25°C και ο 0xFFFF σε -0.5°C.
- Η ρουτίνα συμπληρώνεται από μια εντολή που επιστρέφει την τιμή 0x8000 σε περίπτωση που δεν υπάρχει συσκευή συνδεδεμένη στον ακροδέκτη PD4.

Το πρόγραμμα πρέπει να έχει παρόμοια δομή με την assembly, δηλ. οι συναρτήσεις της C να αντιστοιχούν υποχρεωτικά στις ρουτίνες που σας δίνονται.

Αντί για τον αισθητήρα DS1820 μπορεί να χρησιμοποιηθεί και ο αισθητήρας DS18B20 με τη διαφορά ότι στην προεπιλεγμένη ρύθμιση της λειτουργίας του DS18B20, η τιμή της θερμοκρασίας μετριέται με ακρίβεια 0.0625 °C και δίνεται σε μορφή συμπληρώματος ως προς 2 σύμφωνα με τον παρακάτω πίνακα:

LS BYTE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
MS BYTE	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
S	S	S	S	S	S	2 ⁶	2 ⁵	2 ⁴

S = SIGN

Για παράδειγμα, ο αριθμός 0x0190 αντιστοιχεί σε θερμοκρασία 25°C και ο 0xFFFF σε -0.5°C.

Ζήτημα 7.2 Γράψτε πρόγραμμα σε C που με την χρήση της προηγούμενης ρουτίνας να απεικονίζει τη θερμοκρασία σε °C στο LCD display σε δεκαδική τιμή τριών ψηφίων με το πρόσημο (-55°C έως +125°C).

Επίσης στην περίπτωση που δεν υπάρχει συσκευή συνδεδεμένη να εμφανίζει το μήνυμα “**NO Device**”.



8^η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικρούπολογιστών"
Συνδυαστική/Επαναληπτική άσκηση – Εφαρμογή Internet of Things (στο ntuAboard)
Εξέταση – Επίδειξη: Παρασκευή 12/12/2025

Προθεσμία για παράδοση Έκθεσης: Τρίτη 16/12/2025 23:59

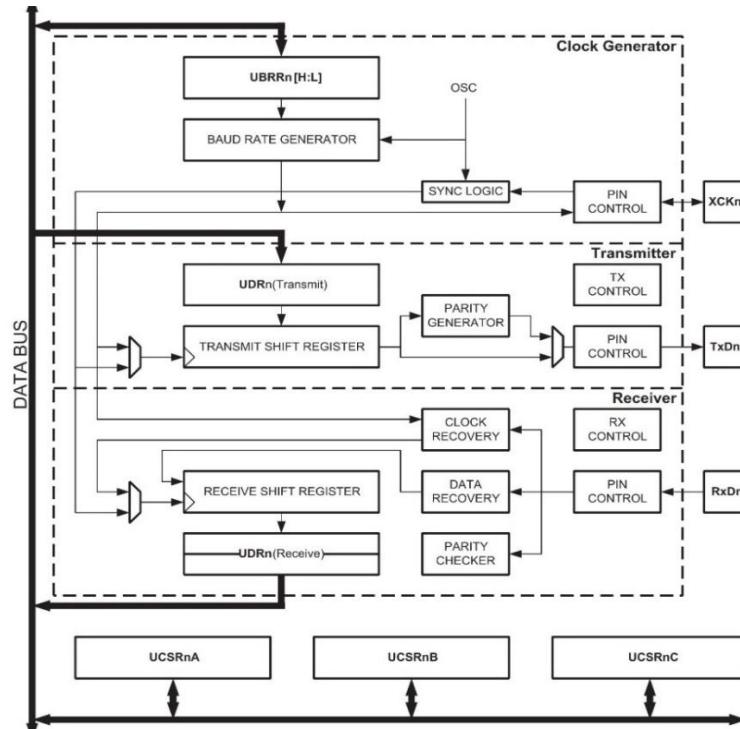
Ασύγχρονη σειριακή επικοινωνία UART

Ο ATmega328PB είναι εξοπλισμένος με δυο μονάδες USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter) οι οποίες μπορούν να χρησιμοποιηθούν για Σύγχρονη αλλά και για Ασύγχρονη επικοινωνία. Στο πλαίσιο αυτής της άσκησης θα μελετηθεί ο Ασύγχρονος τρόπος επικοινωνίας (UART) και η μονάδα USART1. Οι δυο μονάδες είναι ισοδύναμες και για αυτό υπάρχει σε όλα τα σχηματικά το γράμμα n, όπου n=0 για την πρώτη μονάδα και n=1 για την δεύτερη.

Η μονάδα UART μπορεί να μεταδώσει 30 συνδυασμούς πλαισίου:

- 1 bit εκκίνησης
- 5,6,7,8 ή 9 bits
- Ένα ή δύο bit λήξης
- Καμία, άρτια ή περιττή ισοτιμία (parity)

Η βασική δομή φαίνεται στο παρακάτω σχήμα:

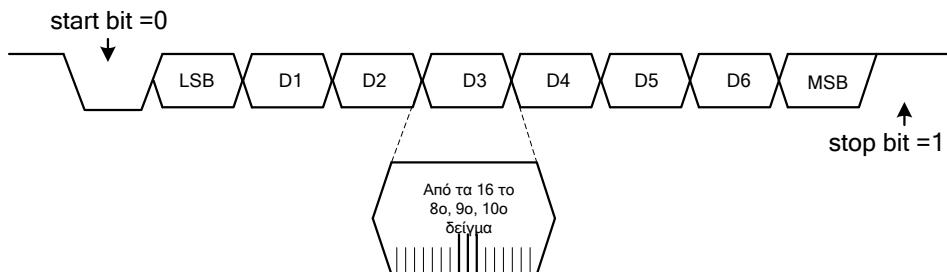


Σχήμα 8.1. Κυκλωματικό διάγραμμα Μονάδας UART.

Ο ακροδέκτης XCKn (Transfer Clock) χρησιμοποιείται μόνο στο σύγχρονο τρόπο επικοινωνίας. Στο ρυθμό μετάδοσης υπάρχει διαίρεση με το 16. Ο πομπός χρησιμοποιεί την διαιρεμένη συχνότητα για την αποστολή ενώ ο δέκτης χρησιμοποιεί το 16πλάσιο της συχνότητας αυτής ώστε να δειγματοληπτεί το σήμα. Από τα δείγματα που λήφθηκαν χρησιμοποιούνται το 8°, το 9° και το 10° και με τον κανόνα της πλειοψηφίας αποφασίζεται αν λήφθηκε λογικό 1, λογικό 0 ή αν πρόκειται για θόρυβο στην περίπτωση που αναμένεται start bit (που πρέπει οπωσδήποτε να είναι 0).

Όταν αναγνωριστεί το start bit πραγματοποιείται ένας "συγχρονισμός" του δέκτη με τον πομπό, αφού αναγνωρίζεται η μετάβαση από λογικό 1 σε λογικό 0 και στη συνέχεια το επόμενο bit θα καταφθάσει σε συγκεκριμένο χρόνο (1/BAUD).

Ο ρυθμός μετάδοσης, ο αριθμός των bit που αναμένονται, ο αριθμός των stop bit και το είδος του parity bit πρέπει να είναι όλα γνωστά στον δέκτη για να λειτουργήσει σωστά η επικοινωνία.



Σχήμα 8.2 Δειγματοληψία δεδομένου

Περισσότερες πληροφορίες σχετικά με την λειτουργία των καταχωρητών αλλά και γενικότερα της UART μπορείτε να βρείτε στο datasheet [ATmega328PB](#).

Καταχωρητές

Καταχωρητής δεδομένων (UDRn)

Bit	7	6	5	4	3	2	1	0	UDRn (Read)	UDRn (Write)
	RXB[7:0]									
	TXB[7:0]									
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	UDRn (Read)	UDRn (Write)
Initial Value	0	0	0	0	0	0	0	0		

Από το σχήμα φαίνεται ότι πομπός και δέκτης μοιράζονται τον καταχωρητή δεδομένων UDRn. Στην πραγματικότητα όμως πρόκειται για δύο διαφορετικούς καταχωρητές που έχουν κοινή φυσική διεύθυνση 0xC6 (UDR0). Ο ένας είναι ο καταχωρητής δεδομένων εκπομπής όπου μεταφέρονται τα δεδομένα που γράφουμε στη διεύθυνση 0xC6. Ο άλλος είναι ο καταχωρητής δεδομένων λήψης όπου μεταφέρονται τα δεδομένα που διαβάζουμε από τη διεύθυνση 0xC6. Ο καταχωρητής δεδομένων λήψης είναι δομημένος σε FIFO 2 επιπέδων και η κατάσταση του αλλάζει σε κάθε προσπέλαση.

Δεν πρέπει να χρησιμοποιούνται οι εντολές SBI και CBI, διότι κάνουν αυτόματη ανάγνωση-τροποποίηση-εγγραφή στον καταχωρητή UDRn. Το μέρος ανάγνωσης της εντολής θα τραβήξει δεδομένα από τον FIFO καταχωρητή και θα χαθούν δεδομένα λήψης.

Χρειάζεται ιδιαίτερη προσοχή στη χρήση των SBIC και SBIS αφού και αυτές κάνουν ανάγνωση του καταχωρητή για να ελέγξουν τα bits(Μετακινούν το FIFO).

Καταχωρητής κατάστασης (USCRnA)

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

RXCn: Αν είναι λογικό 1 τότε υπάρχουν δεδομένα προς ανάγνωση. Μετά την ανάγνωση τους η τιμή μηδενίζεται.

TXCn: Τίθεται σε λογικό 1 όταν έχει ολοκληρωθεί η ολίσθηση όλων των δεδομένων και δεν υπάρχουν καινούρια δεδομένα στον καταχωρητή UDRn.

UDREn: Αν είναι λογικό 1 ο καταχωρητής αποστολής δεδομένων UDRn είναι έτοιμος να δεχθεί νέα δεδομένα.

Καταχωρητής ελέγχου (USCRnB)

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXENn: Αν είναι λογικό 1 ενεργοποιείται το τμήμα λήψης της μονάδας UART

TXENn: Αν είναι λογικό 1 ενεργοποιείται το τμήμα εκπομπής της μονάδας UART

Καταχωρητής ελέγχου (USCRnC)

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USB8n	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

UMSELn1:0: Επιλογή του τρόπου λειτουργίας της USART. 00 για Ασύγχρονη λειτουργία.

UCSZn1:0: Τα bit UCSZn1:0 σε συνδυασμό με το bit UCSZn2 του καταχωρητή USCRnB καθορίζουν των αριθμό των bit δεδομένων (Character Size) που ανταλλάσσονται πομπός και δέκτης. Συγκεκριμένα:

Table 66. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Καταχωρητής του ρυθμού μετάδοσης (UBRRn)

Bit	15	14	13	12	11	10	9	8	UBRRnH
	-	-	-	-	UBRRn[11:8]			UBRRnL	
	UBRRn[7:0]								
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UBRR11:0: Πρόκειται για έναν καταχωρητή 12-bit που περιέχει το ρυθμό μετάδοσης. Ο καταχωρητής UBRRnH περιέχει τα 4 MSB, και ο UBRRnL περιλαμβάνει τα 8 LSB. Εγγραφή στον UBRRnL ανανεώνει άμεσα τον ρυθμό και θα προκαλέσει αλλοίωση των δεδομένων σε περίπτωση που γίνει κατά τη διάρκεια εκπομπής.

Για τον υπολογισμό του ρυθμού μετάδοσης BAUD από την τιμή του καταχωρητή ισχύει ο τύπος

$$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$$

όπου f_{osc} η συχνότητα λειτουργίας του μικροελεγκτή. Λύνοντας ως προς UBRR έχουμε:

$$UBRR = \frac{f_{osc}}{16BAUD} - 1$$

Για να έχουμε BAUD=9600 και αφού ο μικροελεγκτής μας λειτουργεί στα 16MHz πρέπει UBRR=103.1667. Στρογγυλοποιούμε στο 103, εισάγοντας ένα σφάλμα 0,16%.

Για να αρχικοποιήσουμε την USART μπορούμε να χρησιμοποιήσουμε την εξής ρουτίνα:

```
/* Routine: usart_init
Description:
This routine initializes the
uart as shown below.
----- INITIALIZATIONS -----

Baud rate: 9600 (Fck= 8MH)
Asynchronous mode
Transmitter on
Reciever on
Communication parameters: 8 Data ,1 Stop, no Parity
-----
parameters: ubrr to control the BAUD.
return value: None.*/

void usart_init(unsigned int ubrr){
    UCSR0A=0;
    UCSR0B=(1<<RXEN0)|(1<<TXEN0);
    UBRR0H=(unsigned char)(ubrr>>8);
    UBRR0L=(unsigned char)ubrr;
    UCSR0C=(3 << UCSZ00);
    return;
}
```

```

/* Routine: usart_transmit
Description:
This routine sends a byte of data
using usart.
parameters:
data: the byte to be transmitted
return value: None. */

void usart_transmit(uint8_t data){
    while(!(UCSR0A&(1<<UDRE0)));
    UDR0=data;
}

/* Routine: usart_receive
Description:
This routine receives a byte of data
from usart.
parameters: None.
return value: the received byte */

uint8_t usart_receive(){
    while(!(UCSR0A&(1<<RXC0)));
    return UDR0;
}

```

Σύντομη εισαγωγή στο Internet of Things

Η ραγδαία ανάπτυξη του Internet of Things (IoT) έχει επιφέρει σημαντική αύξηση των συσκευών που είναι συνδεδεμένες στο διαδίκτυο. Ο αριθμός αυτός εξακολουθεί να αυξάνεται με ταχύτατους ρυθμούς. Συνέπεια αυτού είναι η παραγωγή τεράστιου όγκου πληροφοριών προς επεξεργασία.

Τι είναι οι Συσκευές IoT:

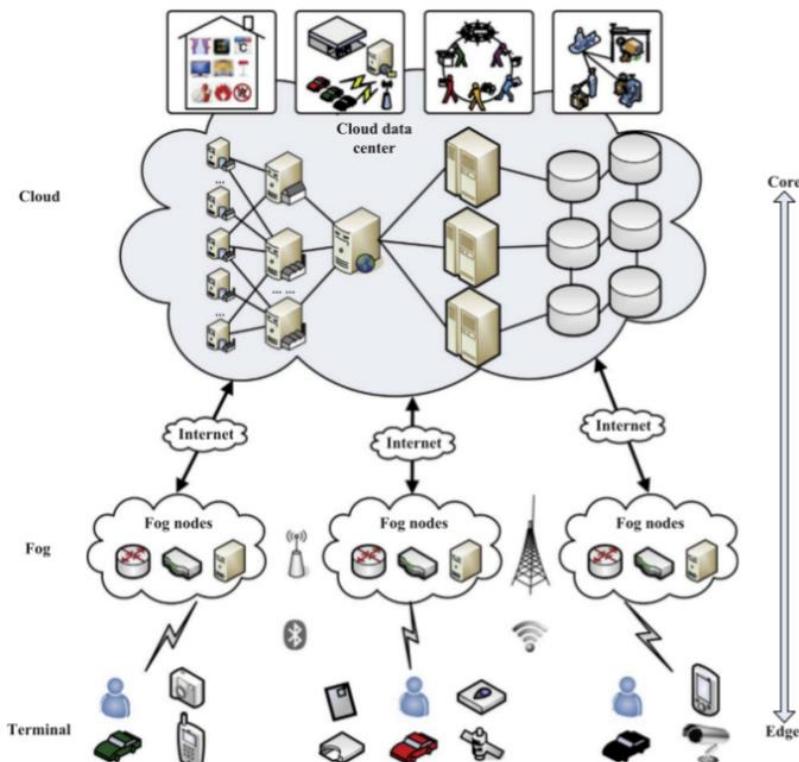
Ως συσκευές IoT ορίζουμε κάθε είδους φυσική συσκευή, οχήματα, οικιακές συσκευές και άλλα αντικείμενα, στα οποία υπάρχουν ενσωματωμένα αισθητήρες, λογισμικό, ενεργοποιητές και σύνδεση στο διαδίκτυο, που τους επιτρέπει να επικοινωνούν και να ανταλλάσσουν δεδομένα είτε μεταξύ τους, είτε με το υπόλοιπο διαδίκτυο. Αυτές οι συσκευές έχουν περιορισμένους πόρους σε ό,τι αφορά την εργασία, τη μνήμη και τις δικτυακές τους ικανότητες.

Τι είναι IoT Gateway:

Καθώς δισεκατομύρια συσκευές συνδέονται πλέον στο δίκτυο, το «Gateway» χρειάζεται προκειμένου να συνδέει υπο-δίκτυα διαφορετικών αρχιτεκτονικών και σε διαφορετικά περιβάλλοντα. Στην κλασική δομή του IoT χρησιμοποιείται ως διαμεσολαβητής/γέφυρα μεταξύ των τελικών συσκευών και του Cloud διαμορφώνοντας κατάλληλα τα δεδομένα που στέλνονται και λαμβάνονται. Τα Gateways συνδέουν επίσης στο Cloud υπο-δίκτυα που στο εσωτερικό τους μπορεί να χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας. Τα Gateways εποπτεύουν την διαδικασία επικοινωνίας εντός και εκτός του υπο-δικτύου στο οποίο βρίσκονται. Μπορούν να εκτελούν διεργασίες ελέγχου, ασφάλειας, φιλτραρίσματος των δεδομένων καθώς επίσης να κάνουν σε χαμηλότερο επίπεδο επεξεργασία των δεδομένων που λαμβάνουν (πιο κοντά στις συσκευές και όχι στο Cloud). Τέλος, μεγάλος αριθμός δεδομένων που λαμβάνονται από τους αισθητήρες των υπο-δικτύων IoT, χρειάζεται το Gateway ως σύστημα συντονισμού και ελέγχου της μεταφοράς τους όχι μόνο από μία συσκευή στο Cloud αλλά και από συσκευή σε συσκευή.

Edge / Fog Computing:

Ωστόσο, τα τελευταία χρόνια, λόγω του γεγονότος ότι οι υποδομές του Cloud είναι γεωγραφικά κεντρικοποιημένες και απομακρυσμένες από τους χρήστες, οι εφαρμογές που απαιτούν επεξεργασία σε πραγματικό χρόνο, με χαμηλό latency αδυνατούν να εκτελεστούν επιτυχώς. Αυτό συμβαίνει λόγω μεγάλου χρόνου μεταφοράς των δεδομένων, συμφόρησης του δικτύου και υποβάθμισης της ποιότητας. Επιπλέον, σε εφαρμογές που απαιτείται ιδιωτικότητα των δεδομένων (π.χ. ιατρικές εφαρμογές), αυτή δε μπορεί να διασφαλιστεί απόλυτα σε απομακρυσμένες υποδομές. Αυτοί οι λόγοι οδήγησαν στη δημιουργία του Edge και του Fog Computing, όπου οι απαιτούμενοι υπολογισμοί γίνονται κοντά στις IoT συσκευές. Το σχήμα της αρχιτεκτονικής αυτής παρουσιάζεται στην Εικόνα 1. Έτσι οι χρήστες ανεξαρτητοποιούνται από τις υποδομές του Cloud και, ταυτόχρονα, ενισχύεται η προστασία των προσωπικών τους δεδομένων.



Εικόνα 1 Αρχιτεκτονική Fog και Edge Computing¹

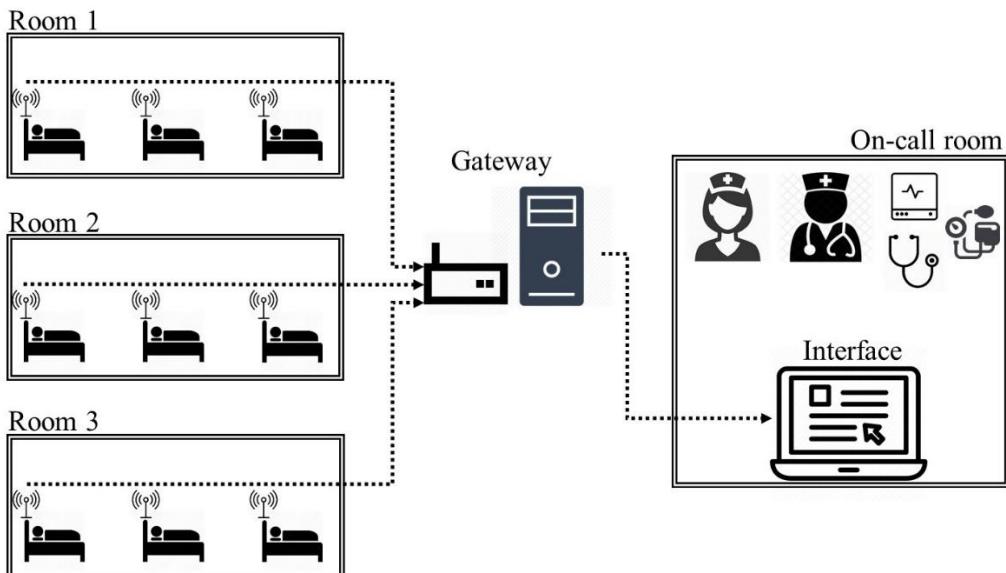
Από τη μία πλευρά, οι συσκευές IoT βρίσκονται κοντά στους καθημερινούς χρήστες και πιο συγκεκριμένα, στο terminal layer της αρχιτεκτονικής που φαίνεται στην Εικόνα 1. Από την άλλη πλευρά o Edge Node (Gateways του Edge Computing, τα οποία συνεισφέρουν στην ανάπτυξη υπηρεσιών Edge και στη παροχή υπολογιστικών, αποθηκευτικών και δικτυακών πόρων στις συσκευές IoT) βρίσκεται στη διεπαφή του Fog layer και του terminal layer. Επιπροσθέτως, οι συσκευές IoT είναι συνδεδεμένες, είτε ενσύρματα, είτα ασύρματα, με το Gateway κόμβο και μπορούν να επικοινωνούν και να ανταλλάσσουν δεδομένα με εκείνον.

Η αρχιτεκτονική του συστήματος που θα μελετήσουμε στα πλαίσια της παρούσας εργασίας αποτελείται από ένα σύνολο n IoT συσκευών (ntuAboard πλακέτες) και μίας συσκευής Gateway. Δεδομένα από αισθητήρες συνδεδεμένους στις συσκευές μπορούν να σταλούν προς επεξεργασία στο Gateway, ενώ τον Gateway μπορεί να στέλνει δεδομένα πίσω στις συσκευές.

¹ P. Hu, S. Dhelim, H. Ning, and T. Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. Journal of Network and Computer Applications, 2017

Περιγραφή της εφαρμογής

Στην παρούσα άσκηση θα μελετηθεί μια εφαρμογή νοσοκομείου. Η Εικόνα 2 δίνει το γενικό σχήμα της εφαρμογής. Το κάθε κρεβάτι (IoT node) έχει συνδεδεμένους αισθητήρες στον ασθενή, τα δεδομένα των οποίων στέλνονται στο Gateway. Το Gateway επεξεργάζεται τα δεδομένα και είτε στέλνει κάποια απάντηση στο IoT node προκειμένου να γίνει κάποια αυτόματη λειτουργία, είτε στέλνει τα δεδομένα στο δωμάτιο των νοσηλευτών και των γιατρών. Τα δεδομένα παρουσιάζονται σε μια οθόνη (Interface) ούτως ώστε να μελετηθούν από το προσωπικό του νοσοκομείου προκειμένου να προβεί στις κινήσεις που απαιτούνται.



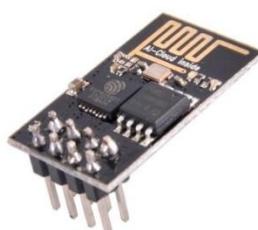
Εικόνα 2 Η εφαρμογή IoT που θα μελετηθεί στην παρούσα άσκηση

Η κάθε συσκευή IoT που βρίσκεται σε κάθε κρεβάτι θα είναι μια πλακέτα ntuAboard, το πρόγραμμα της οποίας θα γραφτεί από την κάθε ομάδα. Ο αριθμός της ομάδας θα είναι ο αριθμός του κρεβατιού. Για λόγους απλότητας ο ένας αισθητήρας που θα χρησιμοποιήσετε θα είναι ο αισθητήρας θερμοκρασίας DS18B20 που μελετήθηκε σε προηγούμενη Εργαστηριακή άσκηση. Για τον δεύτερο αισθητήρα θα χρησιμοποιηθεί ένα ποτενσιόμετρο για να προσδομοίωσει την λειτουργία ενός αισθητήρα κεντρικής φλεβικής πίεσης 0-20 cm H2O.

Το Gateway θα είναι ήδη προγραμματισμένο και το Interface του δωματίου προσωπικού του νοσοκομείου θα παρουσιάζεται στον προβολέα του εργαστηρίου. Αν κάποια ομάδα το επιθυμεί, υπάρχει δυνατότητα να αναπτύξει δικό της πρόγραμμα για το Gateway.

Ο πομποδέκτης WiFi ESP8266:

Ο πομποδέκτης WiFi που θα χρησιμοποιήσουμε για την παραπάνω εφαρμογή είναι ο ESP8266 (Εικόνα 3). Πρόκειται για έναν προγραμματίσιμο πομποδέκτη ο οποίος έχει προγραμματιστεί να δέχεται εντολές μέσω σειριακής και να στέλνει αντίστοιχη απάντηση.



Εικόνα 3 Ο πομποδέκτης WiFi ESP8266

Συγκεκριμένα, κάποιες από τις εντολές που δέχεται φαίνονται στον παρακάτω πίνακα:

command	usage
baudrate: "Your baudrate"	Resets the Serial to the required speed
configuration	Reports your current configuration
connect	Uses the provided SSID and Password to connect to the WiFi network
debug: "true or false"	Every command will send extra debug information if set to true
help	Lists all commands
password: "Your Password"	Password to be used for the connection
payload: [Your payload]	The payload to post to the Host. Must be in brackets
restart	Restarts the ESP
ssid: "Your SSID"	SSID to be used for the connection
transmit	Transmits the specified payload to the url
url: "Your url"	The url to transmit to. Example: http://192.168.1.250:5000/data"

Προκειμένου το ESP8266 να αναγνωρίσει ότι πρέπει να διαβάσει τα δεδομένα της σειριακής, οτιδήποτε στείλετε πρέπει να ξεκινάει με ESP: (π.χ. για σύνδεση θα στείλετε ESP:connect).

Το ESP μπορεί να συνδεθεί στον server σαν client και να στέλνει αιτήματα στον server για να γίνει ανταλλαγή δεδομένων.

Δώστε προσοχή στα εξής:

- 1) Σε όποια εντολή υπάρχουν τα "" πρέπει να τα στέλνετε. Κάθε εντολή που στέλνετε καθώς και κάθε απάντηση που λαμβάνετε θα έχει στο τέλος τον χαρακτήρα αλλαγής γραμμής '\n'.
- 2) Μερικές φορές οι client αποσυνδέονται οπότε να ελέγχετε κάθε φορά με την εντολή connect και μπορείτε σε περίπτωση που αποτύχει να δοκιμάζετε άλλη μια φορά (σε περίπτωση αποτυχίας υπάρχει timeout 20sec όπου το ESP δεν θα αποκρίνεται).
- 3) Μπορείτε στην αρχή του προγράμματος σας να στέλνετε την εντολή restart για να σιγουρευτείτε ότι το ESP βρίσκεται στις default επιλογές του. Μετά την επανεκκίνηση το ESP τυπώνει μια γραμμή με κάποιες πληροφορίες και την γραμμή ESP8266: Waiting for command.
- 4) Το payload που δέχεται ο server πρέπει να είναι σε μορφή json και συγκεκριμένα η εντολή πρέπει να είναι στην μορφή:

```
payload:  
[{"name": "temperature","value": "36.0"}, {"name": "pressure","value": "60.0"}, {"name": "team","value": "5"}, {"name": "status","value": "OK"}]
```

- 5) Η μεταβλητή team είναι απαραίτητη στο payload και σε περίπτωση που δεν την συμπεριλάβετε θα λάβετε ειδικό μήνυμα λάθους από το gateway. Σε περίπτωση που λείπει κάποια από τις άλλες μεταβλητές η έχετε κάνει τυπογραφικό θα συμπληρωθεί με None.
- 6) Τα ονόματα των μεταβλητών στο payload είναι αυστηρά. Σε όποια εντολή υπάρχουν τα "" πρέπει να τα στέλνετε. Κάθε εντολή πρέπει να τελειώνει με τον χαρακτήρα αλλαγής γραμμής '\n'. Τα ssid, password και baudrate(9600) **έχουν ήδη ρυθμιστεί για εσάς**.

Default τιμές:

baudrate	9600
debug	false
password	Microlab_IoT
ssid	Micro_IoT

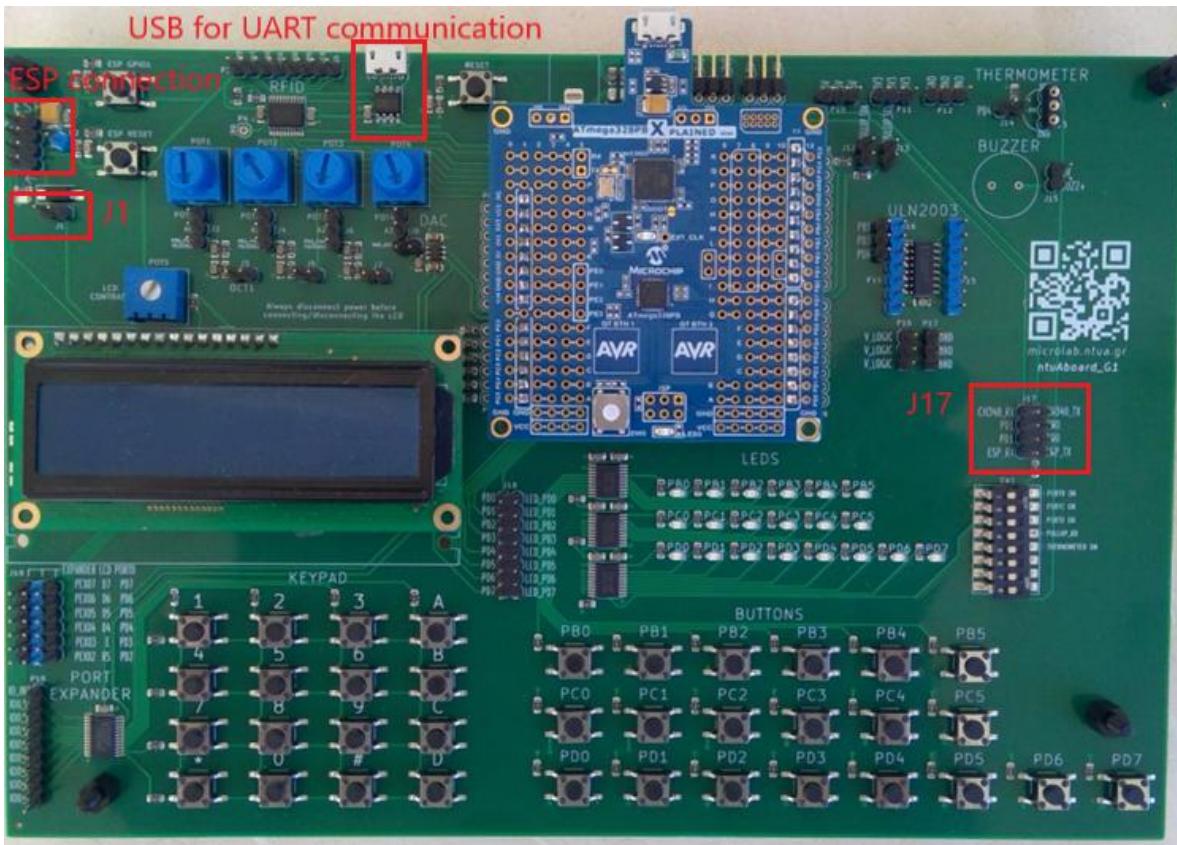
Σας δίνεται παράδειγμα επικοινωνίας με το ESP:

SENDER	MESSAGE
AVR	ESP:connect
ESP	"Success"
AVR	ESP:url:"http://192.168.1.250:5000/data"
ESP	"Success"
AVR	ESP:payload:[{"name": "team", "value": "5"}]
ESP	"Success"
AVR	ESP:transmit
ESP	200 OK

Jumpers

H UART μπορεί μέσω του J17 να συνδεθεί είτε με το ESP8266 είτε με το CH340N το οποίο είναι ένα ολοκληρωμένο υπεύθυνο για την μετατροπή των σημάτων του διαύλου USB σε σήματα σειριακής επικοινωνίας και το αντίστροφο. Αυτό δίνει την δυνατότητα στον υπολογιστή σας να επικοινωνήσει με τη θύρα UART του ATmega328 αν συνδέσετε ένα επιπλέον καλώδιο USB από τον υπολογιστή σας στο αριστερό μέρος του ntuAboard που φαίνεται στην εικόνα. Για την σύνδεση με το ESP8266 χρειάζεται να συνδέσετε το ESP8266 στο ntuAboard, τα jumper στα J17 και J1 όπως φαίνονται στις παρακάτω εικόνες.

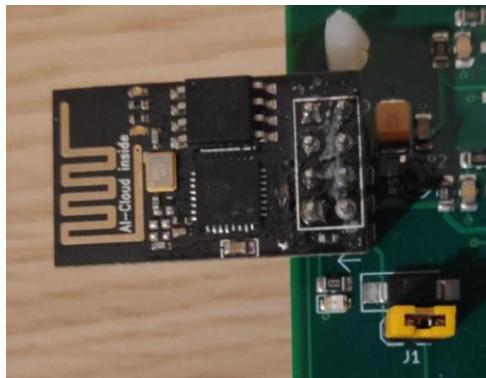
ΠΡΟΣΟΧΗ: Συνδέστε το ESP8266 μόνο με τη φορά που φαίνεται στην εικόνα!



Εικόνα 4 ESP8266 και UART στο ntuAboard



Εικόνα 5 Jumper στο J17 για επικοινωνία με ESP8266 (αριστερά) ή το CH340 (δεξιά)



Εικόνα 6 Σύνδεση ESP8266 και J1 jumper

Μπορείτε να χρησιμοποιήσετε την UART και να προσομοιώνετε εσείς τις απαντήσεις του ESP8266 από τον υπολογιστή για να σιγουρευτείτε ότι λειτουργεί σωστά η σειριακή επικοινωνία. Για την επικοινωνία με τον υπολογιστή σας μπορείτε να χρησιμοποιήσετε οποιοδήποτε πρόγραμμα σειριακής επικοινωνίας επιθυμείτε. Ένα πρόγραμμα είναι το Arduino IDE 2.0.3 που μπορείτε να κατεβάσετε στον υπολογιστή σας από εδώ (<https://www.arduino.cc/en/software>) και αφού το ανοίξετε επιλέξτε Tools->Port και επιλέξτε τη θύρα που είναι συνδεδεμένο το USB σειριακής επικοινωνίας (το αριστερά) και στη συνέχεια Tools->Serial Monitor. Επιλέξτε το σωστό baud από δεξιά κάτω.

Για τα πρώτα ερωτήματα θα χρησιμοποιήσετε την UART και θα προσομοιώνεται εσείς τις απαντήσεις του ESP8266 και μόνο στο τελευταίο ερώτημα θα συνδέσετε το ESP8266 στο ntuAboard.

Τα ζητούμενα της 8^{ης} εργαστηριακής άσκησης

Ζήτημα 8.1 Να γραφεί πρόγραμμα σε C για τον ATmega328 στο ntuAboard το οποίο να στέλνει στην UART κατάλληλη εντολή για να συνδεθεί το ESP8266 στο δίκτυο. Στη συνέχεια, θα διαβάζει από την UART και θα περιμένει την απάντηση του ESP8266 και αν αυτή είναι "Success" θα απεικονίζει στην LCD το μήνυμα 1.Success ενώ αν είναι "Fail" θα τυπώνει στην LCD το μήνυμα 1.Fail. Επαναλάβετε το ίδιο για την εντολή url ώστε να θέσετε το url σε <http://192.168.1.250:5000/data> και αντίστοιχα να εμφανίζεται μηνύματα 2.Success και 2.Fail. Μπορείτε να προσθέσετε μια καθυστέρηση πριν την αποστολή της δεύτερης εντολής για να προλάβετε να δείτε τα μηνύματα στην LCD.

Ζήτημα 8.2 Επεκτείνετε το παραπάνω πρόγραμμα ώστε μετά την εκτύπωση του 2^{ου} μηνύματος για επαρκή χρόνο

(α) να διαβάζει μια μέτρηση από τον αισθητήρα θερμοκρασίας DS18B20 με σκοπό να ενσωματώσετε την τιμή στο payload που θα στείλετε. Προσθέστε στην τιμή του αισθητήρα μια τιμή ώστε το αποτέλεσμα σας να είναι σε συνθήκες θερμοκρασίας δωματίου κοντά στο 36 για να προσομοιάζει πραγματική θερμοκρασία ασθενούς.

(β) να διαβάζει μια μέτρηση από το POT0 και να την μετατρέπετε σε κλίμακα 0-20 cm H2O για να συμπεριφέρεται σαν αισθητήρας κεντρικής φλεβικής πίεσης. Και αυτή η τιμή πρέπει να ενσωματωθεί στο payload.

(γ) να διαμορφώνει το status ως εξής:

1) Αν πατηθεί στο πληκτρολόγιο το πλήκτρο που αντιστοιχεί στο τελευταίο ψηφίο της ομάδας σας τότε το status να γίνεται NURSE CALL. Αν πατηθεί στη συνέχεια η διεση # τότε το status να γίνεται OK εκτός αν συμβαίνει κάτι από τα 2 ή 3.

2) Αν η πίεση είναι πάνω από 12 η κάτω από 4 τότε το status να γίνεται CHECK PRESSURE.

3) Αν η θερμοκρασία είναι κάτω από 34 η πάνω από 37 τότε το status να γίνεται CHECK TEMP.

4) Αν δεν έχει συμβεί κανένα από τα παραπάνω τότε το status να γίνεται OK.

Την τελική τιμή του status ενσωματώστε την στο payload.

Εμφανίστε τις τιμές της θερμοκρασίας και του αισθητήρα κεντρικής φλεβικής πίεσης στην 1^η γραμμή της LCD και το status στην 2^η γραμμή για επαρκή χρόνο.

Ζήτημα 8.3 Επεκτείνετε το παραπάνω πρόγραμμα ώστε να στέλνει στην UART κατάλληλες εντολές για την αποστολή του payload στον server. Αυτές είναι η εντολή payload για την οποία αντίστοιχα να εμφανίζεται μηνύματα 3.Success και 3.Fail και η εντολή transmit. Για την transmit δεν στέλνει το ESP Success ή Fail αλλά στέλνει κατευθείαν την απάντηση του Server. Να τυπώνετε στην οθόνη την απάντηση του server στη μορφή 4. Απάντηση server. Αν όλα έχουν γίνει σωστά πρέπει να λάβετε 200 OK και να δείτε τα στοιχεία που στείλατε στον προβολέα του εργαστηρίου.

Το πρόγραμμα πρέπει να επαναλαμβάνει την λειτουργία του από το 8.1-8.3.