



ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΊΔΡΥΜΑ  
ΑΘΗΝΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ  
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και υλοποίηση συστήματος αυτοματισμού  
για την απομακρυσμένη παρακολούθηση  
περιβαλλοντικών συνθηκών

Θεόδωρος Ελευθέριος ΠΑΝΟΥ  
071045

Επιβλέπουσα καθηγήτρια  
Ιφιγένεια ΦΟΥΝΤΑ

Αθήνα, Νοέμβριος 2014



### **Ευχαριστίες**

Η καθοδήγηση και η εμφύχωση της επιβλέπουσας καθηγήτριας, καθ' όλη τη διάρκεια ανάπτυξης της εργασίας, απετέλεσαν καταλύτης για την πραγμάτωση αυτού του μεγάλου έργου. Τα λόγια αδυνατούν να περιγράψουν επαρκώς τη σπουδαιότητα του ρόλου τής αξιότιμης κυρίας Ιφιγένειας Φουντά.

Σημαντική ήταν η συμβολή και του κυρίου Αθανάσιου Νασιόπουλου για δύσβατες πτυχές της εργασίας που ξεφεύγουν από τα πλαίσια της επιστήμης της Πληροφορικής. Επίσης, ευχαριστώ το προσωπικό του Τμήματος στο σύνολό του καθώς και αυτό έχει συμβάλει, μέσω των σπουδαστικών ετών, στο τρέχον έργο.



## Περίληψη

Η εργασία ασχολείται με την υλοποίηση μίας συσκευής που αυτοματοποιεί την παρακολούθηση συνυθηκών που επιχρωτούν στο υλικό ανοικτού δοχείου το οποίο αυτή περιβάλλει. Η πρόσβαση στη συσκευή πραγματοποιείται απομακρυσμένα με χρήση πρωτοκόλλου HTTP. Η υλοποίηση στηρίζεται στη χρήση μίας πλακέτας Arduino Uno rev3 και ενός μικροελεγκτή Atmel AVR ATmega328 που αυτή περιέχει. Ο προγραμματισμός του μικροελεγκτή γίνεται δια μέσω λογισμικού Boot loader και διασύνδεσης USB. Η ανάπτυξη του λογισμικού υποβοηθείται μόνο από την AVR GNU συλλογή μεταγλωττιστών avr-gcc και τη βιβλιοθήκη προγραμματισμού AVR Libc. Κρίσιμες δυνατότητες που ο συγκεκριμένος μικροελεγκτής στερείται, όπως ρολόι πραγματικού χρόνου, δικτυωσή διασύνδεση και επιπρόσθετο αποθηκευτικό χώρο, παρέχονται από εξωτερικά ολοκληρωμένα χυκλώματα. Η συσκευή ρυθμίζεται σε σχέση με παραμέτρους βιοηθητικών λειτουργιών της καθώς και της διαδικασίας παρακολούθησης. Όλες οι ρυθμίσεις διατηρούνται μεταξύ διακοπών τροφοδοσίας και είναι δυνατό να επαναφερθούν στις εργοστασιακές της ρυθμίσεις με μηχανικό τρόπο. Για τη διασύνδεση της με το χρήστη, υλοποιείται λογισμικό διαχομιστή HTTP μέσω του οποίου επιτυγχάνεται η ανάκτηση πόρων, οι οποίοι, σε συνδυασμό με παρεχόμενες μεθόδους του πρωτοκόλλου HTTP, επιτρέπουν πρόσβαση σε αυτήν. Η μορφή των αναπαραστάσεων των πόρων γίνεται σε JSON και προορίζεται για αξιοποίηση από εξωτερικές εφαρμογές. Επιπλέον, υποστηρίζονται πόροι σε HTML, CSS, JavaScript και PNG για χρήση της συσκευής μέσω λογισμικού πλογγησης. Το κεντρικό σημείο ενδιαφέροντος της εργασίας έγκειται στην υλοποίηση ενός ολοκληρωμένου συστήματος που εμπλέκει πληθώρα τεχνολογικών πεδίων.

## **Abstract**

The project's aims is a device that automates monitoring the conditions of the material within a container. Access to the device is provided remotely via the HTTP protocol. The core of the implementation is an Arduino Uno rev3 board and an Atmel AVR ATmega328 microcontroller contained therein. The microcontroller is programmed via Boot loader software on a USB interface. The development of the firmware is based solely on the GNU AVR toolchain and AVR Libc library. Important functionality not inherently present on the microcontroller, such as real-time clock, network interfacing and additional storage, is provided by external integrated circuits. The device may be configured as far as monitoring parameters are concerned as well as supplementary functions it provides. A mechanism is provided so the configuration is unaffected from power outage, while allowing to manually reset it to its default factory settings, if needed. The user interface is based on a rudimentary HTTP server; HTTP resources in combination with HTTP methods provide the fundamental mechanism for remote access. The resources are represented in JSON format, mostly intended for external software systems. A few additional resources are provided in HTML, CSS, JavaScript and PNG intended to be viewed in a web browser. The primary reason for undertaking such a project lies in the desire to implement a complete system that incorporates various technological fields.

# Περιεχόμενα

<b>1</b>	<b>Εισαγωγή</b>	<b>1</b>
<b>2</b>	<b>Περιγραφή υλοποίησης</b>	<b>3</b>
2.1	Περιβάλλον ανάπτυξης . . . . .	3
2.1.1	Πλατφόρμα Arduino . . . . .	3
2.1.2	Λοιπό λογισμικό ανάπτυξης . . . . .	8
2.2	Μικροελεγκτής . . . . .	11
2.2.1	Μνήμη προγράμματος . . . . .	11
2.2.2	Καθήκοντα μικροελεγκτή . . . . .	12
2.2.3	Κατάσταση χαμηλής κατανάλωσης . . . . .	13
2.3	Η υλοποίηση συνοπτικά . . . . .	14
2.3.1	Κατασκευή . . . . .	14
2.3.2	Κωδικοποίηση κίνησης . . . . .	14
2.3.3	Υποσύστημα κίνησης . . . . .	15
2.3.4	Μετρήσεις . . . . .	15
2.3.5	Επικοινωνία με συσκευή . . . . .	16
<b>3</b>	<b>Λειτουργίες υποδομής</b>	<b>17</b>
3.1	Μνήμες αποθήκευσης . . . . .	19
3.1.1	Αρχεία διεπαφής . . . . .	19
3.1.2	Εφεδρική μνήμη . . . . .	24
3.2	Ημερολόγιο . . . . .	25
3.2.1	Δομή εγγραφής . . . . .	26
3.2.2	Ομάδα εγγραφών . . . . .	27
3.2.3	Οργάνωση εγγραφών . . . . .	28
3.3	Ωρα συστήματος . . . . .	30
3.3.1	Μνήμη RAM . . . . .	31
3.4	Εργασία . . . . .	31
3.4.1	Δειγματοληψία . . . . .	32

3.4.2	Μέτρηση θερμοκρασίας . . . . .	37
3.5	Δίαυλοι επικοινωνίας ολοκληρωμένων	41
3.5.1	Δίαυλος 1-Wire . . . . .	42
3.5.2	Δίαυλος I <sup>2</sup> C (TWI) . . . . .	46
3.5.3	Δίαυλος SPI . . . . .	47
<b>4</b>	<b>Κατασκευή</b>	<b>49</b>
4.1	Μηχανή CNC . . . . .	50
4.1.1	Διατάξεις CNC . . . . .	51
4.1.2	Τυπόδειγμα κατασκευής . . . . .	53
4.2	Σύστημα κατασκευής . . . . .	54
4.2.1	Οδηγοί . . . . .	55
4.2.2	Τροχοί . . . . .	56
4.3	Βάση . . . . .	58
4.4	Άξονες κίνησης . . . . .	59
4.4.1	Άξονας Y . . . . .	59
4.4.2	Άξονας X . . . . .	59
4.4.3	Άξονας Z . . . . .	61
4.5	Τοποθέτηση κινητήρων . . . . .	63
<b>5</b>	<b>Κωδικοποιητής κίνησης</b>	<b>67</b>
5.1	Οπτικοί κωδικοποιητές . . . . .	69
5.1.1	Αναγνώριση θέσης . . . . .	69
5.1.2	Διάταξη στοιχείων . . . . .	70
5.2	Ανωκλαστικός αισθητήρας TCRT5000 . . . . .	71
5.2.1	Συντελεστής σύζευξης . . . . .	71
5.2.2	Λοιπές παράμετροι . . . . .	78
5.2.3	Υπολογισμοί . . . . .	80
<b>6</b>	<b>Τυποσύστημα κίνησης</b>	<b>85</b>
6.1	Συντεταγμένες και Λειτουργικό εύρος . . . . .	87
6.2	Παραγωγή κίνησης . . . . .	88
6.2.1	Γεννήτρια PWM . . . . .	89
6.3	Δρομολόγηση σημάτων . . . . .	92
6.3.1	Διεκπεραίωση . . . . .	93
6.4	Εγκλωβισμός και επανάκαμψη . . . . .	97
6.4.1	Προσαρμογή στις ανάγκες της υλοποίησης . . . . .	99
6.4.2	Αναγγελία εγκλωβισμού . . . . .	100
6.5	Παλινόστηση . . . . .	101

<b>7 Διαδικτύωση</b>	<b>103</b>
7.1 Το ολοκληρωμένο δικτύωσης W5100 . . . . .	105
7.1.1 Πλακέτα WIZ811MJ . . . . .	106
7.1.2 Διασύνδεση με μικροελεγκτή . . . . .	107
7.1.3 Διευθέτηση W5100 . . . . .	110
7.1.4 Συμπληρωματικό λογισμικό W5100 . . . . .	115
7.2 Υποσύστημα διαχομιστή HTTP . . . . .	119
7.2.1 Κορμός διαχομιστή . . . . .	120
7.2.2 Αναλυτής HTTP . . . . .	124
7.2.3 Πόροι διαχομιστή . . . . .	126
7.2.4 Μονάδα αναπαράστασης πόρων . . . . .	129
7.3 Πόροι υλοποίησης . . . . .	133
7.3.1 Πόρος /configuration . . . . .	134
7.3.2 Πόρος /coordinates . . . . .	136
7.3.3 Πόρος /measurement . . . . .	137
7.3.4 Πόροι αρχείων . . . . .	139
<b>8 Συμπεράσματα</b>	<b>141</b>
8.1 Απρόσμενες συμπεριφορές . . . . .	141
8.2 Βελτιώσεις . . . . .	143
8.3 Εξέλιξη . . . . .	144
<b>Παραρτήματα</b>	
<b>A Συνδεσμολογία</b>	<b>149</b>
<b>B Πηγαίος κώδικας</b>	<b>151</b>
B.1 Μικροελεγκτής . . . . .	151
B.2 Διεπαφή χρήστη («ιστοσελίδα») . . . . .	382
<b>C Φωτογραφία</b>	<b>433</b>



# Κατάλογος σχημάτων

2.1.1 Πλακέτα Arduino Uno. . . . .	4
2.1.2 Η αλυσίδα εργαλείων AVR GCC για τον προγραμματισμό του μικροελεγκτή. . . . .	6
2.2.1 Καθήκοντα του μικροελεγκτή. . . . .	12
3.0.1 Σχέσεις μεταξύ των βασικών μονάδων της υλοποίησης. . . . .	18
3.2.1 Δομή μίας εγγραφής Ημερολογίου. . . . .	26
3.2.2 Αναγωγή ιδεατής θέσης εγγραφής σε φυσική διεύθυνση. . . . .	30
3.3.1 Παράδειγμα εγγραφής και ανάγνωσης από διεύθυνση του RTC. . .	31
3.4.1 Σταδιακή ολοκλήρωση φόρτου εργασίας μετρήσεων. . . . .	33
3.4.2 Στάδια περάτωσης μίας εργασίας μέτρησης. . . . .	34
3.4.3 Αντιστοίχηση ωρών σε κβάντα και υπολογισμός χρονικού διαστήματος. . . . .	36
3.4.4 Πτητική μνήμη (scratchpad) και μνήμη EEPROM του DS18B20. .	38
3.4.5 Μορφή θερμοκρασίας στους καταχωρητές 0 και 1 του DS18B20. .	39
3.4.6 Συναλλαγές 1-Wire για τη δειγματοληψία της θερμοκρασίας. . . .	40
3.5.1 Χρονοθυρίδες εγγραφής και ανάγνωσης στο δίαυλο 1-Wire. . . .	43
3.5.2 Κύκλος συναλλαγής στο δίαυλο 1-Wire. . . . .	44
3.5.3 Χρονισμός δεδομένων SPI με φάση ρολογιού CPHA = 0. . . . .	48
4.0.1 Αναπαράσταση της συσκευής. . . . .	50
4.1.1 Διατάξεις κινητής τραπέζης. . . . .	52
4.1.2 Διατάξεις σταθερής τραπέζης. . . . .	53
4.2.1 Τα τέσσερα μεγέθη οδηγών V-Slot. . . . .	55
4.2.2 Μέλη που αποτελούν τον τροχό. . . . .	56
4.2.3 Πλάκα γεφυρώματος . . . . .	57
4.3.1 Η βάση της συσκευής. . . . .	58
4.4.1 Κινητή τροχαλία και ιμάντας. . . . .	60
4.4.2 Συστατικά μέρη γεφυρώματος. . . . .	60

4.4.3 Αναπαράσταση οδηγού αξόνων Z. . . . .	62
4.5.1 Κανάλι Actobotics και διάταξη οπών. . . . .	63
4.5.2 Διάταξη κινητήρων αξόνων X και Y. . . . .	64
 5.0.1 Επίδραση και ανάδραση μικροελεγκτή και κινητήρα. . . . .	67
5.0.2 Σχηματική απεικόνιση κωδικοποιητή υλοποίησης. . . . .	68
5.1.1 Προσαυξητικός οπτικός κωδικοποιητής με χρήση φωτοδιακόπτη. .	69
5.2.1 Ο ανακλαστικός οπτικός αισθητήρας TCRT5000. . . . .	71
5.2.2 Λειτουργική απόσταση και λειτουργικό διάγραμμα. . . . .	73
5.2.3 Σχέση μεταβολής ρεύματος $I_C$ και συντελεστή ανάκλασης. . . .	74
5.2.4 Σχέση Λειτουργικής απόστασης και Διαστήματος εναλλαγής. . .	75
5.2.5 Σχέση Συχνότητας αποκοπής και Αντίστασης φόρτου. . . . .	76
5.2.6 Σχέση θερμοκρασίας και Σχετικού λόγου μεταφοράς ρεύματος. .	77
5.2.7 Σχέση Ρεύματος ορθής φοράς και Λόγου μεταφοράς ρεύματος. .	78
5.2.8 Σχέση θερμοκρασίας και Ρεύματος ηρεμίας. . . . .	79
5.2.9 Σχέση Κατανάλωσης ισχύος και Θερμοκρασίας. . . . .	80
 6.0.1 Υποσύστημα κίνησης. . . . .	85
6.0.2 Ανάλυση υποσυστήματος κίνησης. . . . .	86
6.1.1 Συντεταγμένες και Λειτουργικό εύρος. . . . .	88
6.3.1 Δρομολόγηση σήματος στους κινητήρες. . . . .	93
6.3.2 Αλληλουχία κύκλων μετατόπισης. . . . .	96
6.3.3 Λογική σύνδεση μετρητή βημάτων και κινητήρων. . . . .	97
6.4.1 Συνδεσμολογία εγκλωβισμού και επανάκαμψης κινητήρα σε ένα όχρο.	98
6.4.2 Πλήρης συνδεσμολογία εγκλωβισμού και επανάκαμψης. . . . .	99
 7.0.1 Επικοινωνία χρήστη και συσκευής μέσω πρωτοκόλλων διαδικτύωσης. . . . .	103
7.0.2 Διαμοιρασμός αρμοδιοτήτων μεταξύ μικροελεγκτή και W5100. . .	104
7.0.3 Μέρη που απαρτίζουν τη μονάδα διαδικτύωσης. . . . .	106
7.1.1 Πλακέτα WIZ811MJ και διάταξη των ακροδεκτών του W5100. .	107
7.1.2 Διασύνδεση μικροελεγκτή και W5100. . . . .	109
7.1.3 Δομή εντολής W5100 μέσω SPI. . . . .	109
7.1.4 Καταχωρητές μεγέθους, RMSR και TMSR. . . . .	111
7.1.5 Καταχωρητές κατάστασης και διακοπών, IR και IMR. . . . .	111
7.1.6 Μνήμη εισερχομένων για κάποιο Socket n. . . . .	113
7.1.7 Γράφος καταστάσεων Socket διακομιστή HTTP. . . . .	114
7.1.8 Ροή χαρακτήρων από τη μνήμη εισερχομένων Socket. . . . .	116

7.1.9 Λεξική ανάλυση ροής Socket. . . . .	118
7.2.1 Αίτημα και απόκριση μεταξύ αιτούσας και οντότητας προμηθευτή. .	119
7.2.2 Διασύνδεση μεταξύ βασικών μονάδων του υποσυστήματος διακομιστή HTTP. . . . .	121
7.2.3 Μηχανισμός ανασύνθεσης τεμαχισμένου σώματος μηνύματος HTTP. .	123
7.2.4 Περιγραφή πόρων διακομιστή. . . . .	127
7.2.5 Δομή αναπαράστασης ερωτήματος URL. . . . .	128
7.2.6 Διασύνδεση ρουτίνας περάτωσης και αναλυτή αναπαράστασης. . .	131
7.3.1 Πόροι υλοποίησης και οι μονάδες που χρησιμοποιούν. . . . .	134



# Κατάλογος πινάκων

2.3.1 Εργοστασιακές ρυθμίσεις δικτύωσης της συσκευής.	16
3.1.1 . . . . .	21
3.1.2 Επίπεδα προστασίας από εγγραφή.	22
5.2.1 Σχετική απόδοση διάφορων αναλαστικών υλικών.	72
6.2.1 Μέρος ρυθμίσεων PCPWM των Timer/Counter0 και 1.	91
6.2.2 Κύκλοι εργασίας και τιμές OCR1x.	92



# Κεφάλαιο 1

## Εισαγωγή

Η εργασία αναλαμβάνει τη μελέτη, το σχεδιασμό και την υλοποίηση ενός συστήματος αυτοματισμού για την απομακρυσμένη παρακολούθηση περιβαλλοντικών συνθηκών.

Το αντικείμενο που επιλέγεται να υποστηριχθεί αντλεί έμπνευση από συστήματα κομποστοποίησης. Η κομποστοποίηση είναι μία διαδικασία που σκοπό έχει τη μετατροπή αγροτικών και οικιακών υπολειμμάτων σε πλούσια οργανική μάζα. Μέσω αυτής, σημειώνονται αρκετά οφέλη για το περιβάλλον και το κοινωνικό σύνολο, όπως, μείωση όγκου απορριμάτων, εμπλουτισμός εδάφους, ανάληψη ευθύνης απέναντι στα υποπροϊόντα της ανθρώπινης δραστηριότητας.

Στο πλαίσιο της υλοποίησης επιλέγεται η παρακολούθηση των συνθηκών ενός υποθετικού δοχείου στο οποίο εναποτίθενται υλικά τα οποία προορίζονται για αποσύνθεση μέσω ψυχρής κομποστοποίησης που, για την ακρίβεια, κάνει χρήση γαιοσκώληκα. Σε ένα τέτοιο σύστημα, πρέπει να παρακολουθούνται, κυρίως, η θερμοκρασία, η υγρασία και η οξύτητα.

Ωστόσο, από τις τρεις αυτές βασικές μεταβλητές, πραγματοποιείται ενδεικτική υλοποίηση παρακολούθησης μόνο της θερμοκρασίας. Προς επίτευξη αυτού, η συσκευή διαθέτει ένα ειδικό εξάρτημα που μπορεί να διανύει την επιφάνεια του δοχείου και να εισχωρεί κατακόρυφα σε αυτό για τη λήψη μετρήσεων.

Τυπικά, η συσκευή ξοδεύει το μεγαλύτερο μέρος της ημέρας σε κατάσταση χαμηλής κατανάλωσης ισχύος, παρότι συνδεδεμένη στην κεντρική τροφοδοσία. Από αυτήν την κατάσταση, αφυπνίζεται αυτόματα για την πραγματοποίηση μετρήσεων του υλικού ή για την απόκριση στις προτροπές του χρήστη.

Σε σχέση με τις αυτόματες μετρήσεις, η συσκευή ρυθμίζεται για τις διαστάσεις του δοχείου που της προσαρτάται, για το χρονικό διάστημα που είναι επιθυμητό να παρεμβάλλεται μεταξύ μετρήσεων καθώς και για το πλήθος μετρήσεων που πραγματοποιούνται κάθε φορά που αφυπνίζεται για το σκοπό αυτό. Οι θέσεις στις

οποίες πραγματοποιούνται μετρήσεις επιλέγονται τυχαία μέσα στο χώρο του δοχείου, ώστε να παρέχεται μία εικόνα για τη συνολική κατάσταση του υλικού. Οι 90 πλέον πρόσφατες πραγματοποιηθείσες μετρήσεις αποθηκεύονται σε μνήμη της συσκευής και φέρουν πληροφορία της ημερομηνίας/ώρας, της θέσης και της θερμοκρασίας σε αυτήν.

Η τρέχουσα ημερομηνία/ώρα καθορίζεται από το χρήστη μία φορά και τηρείται από κύκλωμά της. Για την ακρίβεια, όλες οι ρυθμίσεις της συσκευής διατηρούνται, ακόμα και εάν αυτή αποσυνδεθεί από την τροφοδοσία· χαρακτηριστικό ιδιαίτερα χρήσιμο κατά των διακοπών παροχής ηλεκτρικής ενέργειας.

Μέσα από το λογισμικό περιήγησης της προτίμησής του, δεδομένου ότι αυτό υποστηρίζει στοιχειωδώς HTML5, CSS3 και Javascript (IE8/2006 και μετά), από οποιασδήποτε μορφής υπολογιστή που έχει πρόσβαση στο δίκτυο της συσκευής, ο χρήστης μπορεί να ενημερώνεται για τις τελευταίες μετρήσεις, να εκτελεί κατά παραγγελία μετρήσεις στις θέσεις που επιθυμεί και, σαφώς, να τροποποιεί τις ρυθμίσεις της συσκευής.

Παρότι η σύνδεση του χρήστη με τη συσκευή γίνεται από μία προκαθορισμένη διεύθυνση, επιτρέπεται η αλλαγή αυτής καθώς και των υπόλοιπων ρυθμίσεων δικτύου, ώστε να διευκολύνεται η πρόσβαση σε αυτήν μέσα από το τοπικό δίκτυο του εκάστοτε χρήστη. Ωστόσο, σε περίπτωση που οι ρυθμίσεις αυτές απολεσθούν (κυρίως της διεύθυνσης IP) με αποτέλεσμα να μην δύναται η πρόσβαση στη διεπαφή της συσκευής, παρέχεται η δυνατότητα επαναφοράς της συσκευής στις εργοστασιακές της ρυθμίσεις. Αυτό επιτυγχάνεται με την προσωρινή απομάκρυνση μίας μικρής coin cell μπαταρίας που διαθέτει κατά το διάστημα που βρίσκεται απενεργοποιημένη (καλώδιο τροφοδοσίας αποσυνδεδέμενο).

Ο πιο προχωρημένος χρήστης έχει στη διάθεσή του ένα εύχρηστο προγραμματιστικό API, το οποίο του επιτρέπει να χειρίζεται τη συσκευή μέσω δικών του εφαρμογών. Όλη η λειτουργικότητα που παρέχεται μέσω της διεπαφής που προορίζεται για λογισμικό πλοήγησης, παρέχεται και μέσω του API – για την ακρίβεια, η διεπαφή αποτελεί μία τέτοια εφαρμογή που κάνει χρήση αυτού του API.

Το API χρησιμοποιεί την μορφή JSON για την αναπαράσταση των πόρων και υλοποιείται με την ανταλλαγή μηνυμάτων HTTP και καθιερωμένων μεθόδων (GET, PUT, POST), ενώ διαγνωστικά μηνύματα, όπως «Το αίτημα μετακίνησης της κεφαλής έγινε αποδεκτό· αναμενόμενος χρόνος ολοκλήρωσης 7s», επιστρέφονται με τη μορφή κατάλληλων κωδικών κατάστασης και πεδίων επικεφαλίδας του πρωτοκόλλου HTTP (στο παράδειγμα, 202 και «Retry-After», αντίστοιχα). Οδηγίες χρήσης παρέχονται μέσα από τη διεπαφή της συσκευής.

## Κεφάλαιο 2

# Περιγραφή υλοποίησης

### 2.1 Περιβάλλον ανάπτυξης

Η επιλογή μικροελεγκτή (MCU) αντί μικροεπεξεργαστή (MPU) για την ανάληψη των καυθηκόντων της υλοποίησης θεωρείται δεδομένη. Ως βασικά στοιχεία υπεροχής των μικροελεγκτών χρίνεται η χαμηλότερη κατανάλωση ισχύος, η έλλειψη ανάγκης σύνδεσης και τροφοδοσίας εξωτερικής κύριας και δευτερεύουσας μνήμης, και ο λιγότερος (φυσικός) χώρος που καταλαμβάνουν (Gaillard και Eieland, 2013, σσ. 1–2). Αυτά βέβαια παρέχονται εις βάρος χαμηλότερης επεξεργαστικής ισχύος και λιγότερης μνήμης που, σε αντίθεση με τους μικροεπεξεργαστές που υποστηρίζουν μεγέθη της τάξης των GiB, οι μικροελεγκτές περιορίζονται σε μερικά MiB.

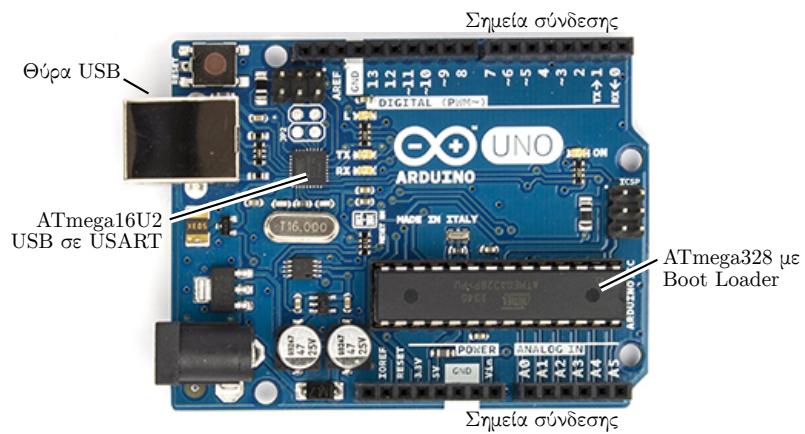
Βέβαια, παρότι ένας μικροελεγκτής είναι πιο συμπαγής, απαιτούνται, τελικά, ορισμένα επιπρόσθετα στοιχεία όπως, για παράδειγμα, για τη ρύθμιση και σταθεροποίηση της τάσης που αυτός λαμβάνει. Επιπλέον, τίθενται ζητήματα που αφορούν τον προγραμματισμό του καυθώς και τη (φυσική) διασύνδεση με τα οποιαδήποτε εξαρτήματα και εξωτερικά ολοκληρωμένα που πρόκειται να χρησιμοποιηθούν.

#### 2.1.1 Πλατφόρμα Arduino

Οι πλακέτες Arduino επιλύουν τα παραπάνω ζητήματα και παρέχουν μία έτοιμη λειτουργική μονάδα που επιτρέπει την άμεση ενασχόληση με τη δημιουργία του πρότυπου λογισμικού και εκείνων των συνδέσεων με ηλεκτρονικά στοιχεία που απαιτούνται στο πλαίσιο αυτού.

Για την ύπαρξη αρκετής ευελιξίας κατά το σχεδιασμό της υλοποίησης (και λόγω έλλειψης πρότερης εμπειρίας και, επομένως, κρίσης), επιλέγεται η (πλέον πρόσφατη, την περίοδο εκπόνησης) πλακέτα Arduino Uno revision 3, της οποίας ο κε-

Σχήμα 2.1.1: Πλακέτα Arduino Uno.



Βασισμένο. *Arduino Uno R3 Front*. Στο Arduino. Arduino Uno. 2014. url: <http://arduino.cc/en/Main/ArduinoBoardUno>

ντρικός μικροελεγκτής είναι ένας AVR ATmega328P της Atmel. Ο συγκεκριμένος μικροελεγκτής διαθέτει συνολική μνήμη προγράμματος 32KiB, 2KiB χύρια μνήμη, συχνότητα ρολογιού που φτάνει τα 20MHz (που, αστόσο, λόγω της πλακέτας, περιορίζεται στα 16MHz) και πληθώρα δυνατοτήτων (Atmel, 2013, σ. 1. Arduino, 2014b). Επιπρόσθετα, η πλακέτα παρέχει διασύνδεση USB για την επικοινωνία του μικροελεγκτή με τον υπολογιστή, τόσο για τον προγραμματισμό του όσο και για την ανταλλαγή δεδομένων με κάποια άλλη εφαρμογή, για παράδειγμα τερματικό. Στο σχήμα 2.1.1 παρουσιάζεται η πλακέτα Arduino και τα βασικά της μέρη.

Ο μικροελεγκτής της πλακέτας Arduino Uno revision 3 διατίθεται με προ-εγκατεστημένο λογισμικό Boot loader επιτρέποντας, με αυτόν τον τρόπο, τον προγραμματισμό της μνήμης προγράμματος χωρίς τη χρήση ειδικού υλικού, αλλά με την απευθείας σύνδεση της πλακέτα μέσω καλωδίου USB (Arduino, 2014a). Περισσότερα σχετικά με το λογισμικό Boot loader αναφέρονται στην ενότητα Μνήμη προγράμματος σ. 11. Ωστόσο, σημειώνεται ότι η επικοινωνία μέσω USB επιτυγχάνεται με ένα δεύτερο μικροελεγκτή της πλακέτας – ενός ATmega16U2 – του οποίου μοναδικός σκοπός είναι η διασύνδεση του πρωτοκόλλου USB (το οποίο υποστηρίζει εγγενώς) με το κύκλωμα USART που διαθέτει, τόσο ο ίδιος, αλλά, κυρίως, ο ATmega328P – ο κεντρικός μικροελεγκτής της πλακέτας (Arduino, 2014b. Atmel, 2012a, σσ. 148,185. Atmel, 2013, σ. 172). Σαφώς, αυτή η διασύνδεση χρησιμοποιείται για κάθισ επικοινωνία μεταξύ υπολογιστή και πλακέτας μέσω καλωδίου USB, ανεξαρτήτως εάν τα δεδομένα προορίζονται για το λογισμικό Boot loader ή το ίδιο το πρόγραμμα.

Για την περαιτέρω διευκόλυνση της ανάπτυξης του λογισμικού του μικροελεγ-

κτή της πλακέτας, παρέχεται ορισμένο επιπρόσθετο λογισμικό Arduino το οποίο λαμβάνει δύο μορφές. Μία εξ αυτών είναι το περιβάλλον ανάπτυξης Arduino IDE του οποίου οι βασικές λειτουργίες είναι η μεταγλώττιση του πηγαίου κώδικα και η μεταφορά του προγράμματος στο μικροελεγκτή (Arduino, 2014a) μέσω μίας περισσότερο ελκυστικής γραφικής διεπαφής. Μία δεύτερη μορφή λογισμικού είναι οι βιβλιοθήκες Arduino οι οποίες παρέχουν εύχρηστες προγραμματιστικές διεπαφές (API) που εσωτερικά κάνουν χρήση των υποκείμενων κυκλωμάτων του μικροελεγκτή της εκάστοτε πλακέτας (είτε απευθείας, είτε μέσω τρίτου λογισμικού) αποκρύπτοντας, με αυτόν τον τρόπο, τις λεπτομέρειες της υλοποίησης (Arduino, 2014c).

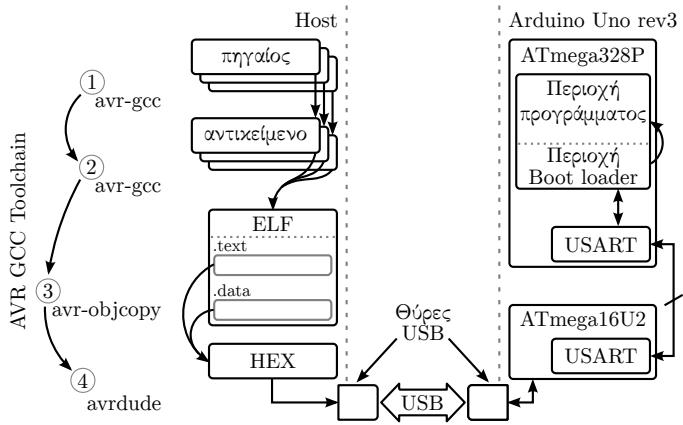
### Εργαλεία προγραμματισμού

Παρότι το IDE και οι βιβλιοθήκες Arduino μπορούν να επιταχύνουν σημαντικά τη διαδικασία ανάπτυξης της υλοποίησης, προτιμάται, αντί αυτών, η απευθείας χρήση καινιερωμένων εργαλείων για τον προγραμματισμό μικροελεγκτών AVR, καθώς έτσι επιδιώκεται να επιτευχθεί σε μεγαλύτερο βάθος κατανόηση της συνολικής διαδικασίας που, ιδανικά, θα επιτρέψει τη μελλοντική ενασχόληση με παρόμοια συστήματα ανεξαρτήτως της πορείας των προϊόντων Arduino. Στο πλαίσιο της υλοποίησης, χρησιμοποιείται μόνο το υλικό, δηλαδή η πλακέτα Arduino για την κάλυψη των βασικών ηλεκτρικών αναγκών, και το λογισμικό Boot loader για τον προγραμματισμό του μικροελεγκτή καθώς και το λογισμικό του ATmega16U2 το οποίο διασυνδέει τον κεντρικό μικροελεγκτή με τη θύρα USB.

Το IDE αντικαθίσταται από έναν οποιοδήποτε επεξεργαστή κειμένου και ο πηγαίος κώδικας διασπάται σε διαφορετικά αρχεία για την καλύτερη οργάνωσή του. Με τον τρόπο αυτό, καθώς η υλοποίηση επεκτείνεται και ο κώδικας της διογκώνεται, ο λογικός διαχωρισμός των λειτουργιών της επιτρέπει το γρηγορότερο εντοπισμό κάποιας συγκεκριμένης υπομονάδας. Επιπλέον, καθώς η μεταγλώττιση μεγάλου όγκου πηγαίου κώδικα μπορεί να αποβεί χρονοβόρα, η ύπαρξη αυτόνομων αρχείων αντικείμενου προγράμματος δημιουργημένων από προηγούμενο κύκλο μεταγλώττισης, επιτρέπει την επαναχρησιμοποίησή τους για την παραγωγή ενός (νέου) συνολικού δυαδικού αρχείου, απαιτώντας μεταγλώττιση μόνο των αρχείων εκείνων που έχουν τροποποιηθεί.

Η μεταγλώττιση των επιμέρους αρχείων πηγαίου κώδικα σε αντικείμενα προγράμματα και η, εν συνεχείᾳ, σύνδεση (ή συνένωσή) τους σε ένα τελικό δυαδικό αρχείο αναλαμβάνεται από την AVR GNU συλλογή μεταγλωττιστών avr-gcc και λοιπών εργαλείων της avr-binutils (όπως ο συνδέτης ld ο οποίος χρησιμοποιείται εσωτερικά από την avr-gcc). Η σύνταξη του κώδικα γίνεται σε γλώσσα C με τη βοήθεια της βιβλιοθήκης AVR Libc.

Σχήμα 2.1.2: Η αλυσίδα εργαλείων AVR GCC για τον προγραμματισμό του μικροελεγκτή.



Το παραγόμενο δυαδικό αρχείο της σύνδεσης είναι μορφής ELF (Executable and Linking Format) η οποία, παρότι παρέχει ανεξαρτησία από επεξεργαστή και αρχιτεκτονική υπολογιστή, είναι, ωστόσο ακατάλληλη για την απευθείας μεταφορά στο μικροελεγκτή AVR (The Santa Cruz Operation, 1997, σ. 47. Savannah GNU, 2012, σ. 346). Απαιτείται, πρώτα, η εξαγωγή των τμημάτων (segment) του κώδικα και των δεδομένων (.text και .data, αντίστοιχα) από το αρχείο ELF σε ένα αρχείο δεκαεξαδικής μορφής (αρχείο HEX) · εργασία η οποία αναλαμβάνεται από το εργαλείο avr-objcopy (Savannah GNU, 2012, σ. 13,346).

Τέλος, το δεκαεξαδικό αρχείο περνάει στο μικροελεγκτή μέσω του εργαλείου avrdude, το οποίο υποστηρίζει προγραμματισμό μικροελεγκτών AVR μέσω διαφόρων προγραμματιστών (συμπεριλαμβανομένου ειδικών εξαρτημάτων όπως οι STK500 και AVRISP mkII της Atmel) και, σαφώς, μέσω Boot loader (Savannah GNU, 2012, σ. 15. Wunsch, 2013).

Σημειώνεται ότι, γενικά, η χρήση Boot loader περιορίζεται στην εγγραφή και ανάγνωση της μνήμης Flash και EEPROM χωρίς να παρέχεται η δυνατότητα εγγραφής των λεγόμενων fuse bit, τα οποία πρόκειται για μη πτητικές θέσεις μνήμης που ρυθμίζουν, με διάφορους τρόπους, τη συμπεριφορά του μικροελεγκτή όπως, για παράδειγμα, το μέγεθος της περιοχής Boot loader (σχετικά με αυτήν, βλ. Μνήμη προγράμματος σ. 11). κάτι τέτοιο είναι δυνατό μόνο μέσω σειριακών ή παράλληλων συνδέσεων προγραμματισμού (Atmel, 2013, σ. 273). Επιπλέον, στην περίπτωση του Boot loader του Arduino Uno revision 3, παρέχεται εγγραφή αποκλειστικά και μόνο για τη μνήμη προγράμματος (μνήμη Flash), κυρίως για τη μείωση του μεγέθους του.

## Εντολές προγραμματισμού

Στο σχήμα 2.1.2 παρουσιάζεται η αλυσίδα (toolchain) των χρησιμοποιούμενων εργαλείων για τη μετάβαση από κάθε στάδιο στο επόμενο με τελικό στόχο τον προγραμματισμό του μικροελεγκτή. Στη βασική τους μορφή, οι εντολές συνοψίζονται παρακάτω.

1. Μετατροπή αρχείων πηγαίου σε αρχεία αντικείμενου προγράμματος για κάθε αρχείο file.c. Τυπικά, η εντολή εκτελείται μόνο για εκείνα τα αρχεία πηγαίου κώδικα που έχουν τροποποιηθεί.

```
avr-gcc -mmcu=atmega328 -Wl,-Map,mcu.map -Os -c file.c
```

**-mmcu** Καθορίζει τον τύπο του μικροελεγκτή ώστε να αναγνωρίζεται το κατάλληλο σύνολο εντολών (instruction set) που πρόκειται να χρησιμοποιηθεί κατά τη συμβολομετάφραση (FSF, 2012).

**-Os** Ενεργοποιεί τις βελτιστοποιήσεις του κώδικα από το μεταγλωττιστή προκειμένου να εξοικονομηθεί χώρος προγράμματος, ακόμα και εις βάρος της ταχύτητας εκτέλεσής του (Optimize for size) (Savannah GNU, 2012, σ. 338. FSF, 2012).

Εκτός του άμεσου πλεονεκτήματος της παραγωγής μικρότερου προγράμματος, η συγκεκριμένη ρύθμιση απαιτείται από τη βιβλιοθήκη AVR Libc προκειμένου να διατίθενται ορισμένες λειτουργίες προσωρινής «παύσης» της εκτέλεσης του προγράμματος (βρόχοι busy-wait) (Savannah GNU, 2012, σ. 328).

**-c** Η συγκεκριμένη επιλογή προκαλεί την παραγωγή αρχείου αντικείμενου προγράμματος όπως αυτό εξάγεται από το συμβολομεταφραστή, αποτρέποντας τη σύνδεσή του σε τελικό πρόγραμμα (Savannah GNU, 2012, σ. 338. FSF, 2012).

2. Σύνδεση των αρχείων αντικείμενου προγράμματος σε αρχείο ELF.

```
avr-gcc -mmcu=atmega328 *.o -o mcu.elf
```

**-mmcu** Ομοίως με την προηγούμενη εντολή.

**-o** Καθορίζει το όνομα του παραγόμενου αρχείου (FSF, 2012).

3. Μετατροπή αρχείου ELF σε δεκαεξαδική μορφή Intel HEX.

```
avr-objcopy -j .text -j .data -O ihex mcu.elf mcu.hex
```

- j Καθορίζει ποια τμήματα του αρχείου ELF θα χρησιμοποιηθούν για τη δημιουργία του αρχείου εξόδου. Στην προκειμένη, εξάγεται ο κώδικας και τα δεδομένα (Savannah GNU, 2012, σ. 346).
- O Καθορίζει τον τύπο του παραγόμενου αρχείου (Savannah GNU, 2012, σ. 346). Στην περίπτωση του μικροελεγκτή AVR, πρόκειται για ένα αρχείο κειμένου ASCII του οποίου τα δεκαεξαδικά ψηφία κωδικοποιούν, ένα προς ένα, τα Byte του αρχικού δυαδικού αρχείου το οποίο αποτελεί το πρόγραμμα που εγγράφεται στη μνήμη Flash (Intel, 1988, σ. 4. Atmel, 2012b, σ. 10).

4. Αποστολή αρχείου HEX στο λογισμικό Boot loader του μικροελεγκτή για τη μετέπειτα εγγραφή του στην περιοχή προγράμματος (βλ. Μνήμη προγράμματος σ. 11).

```
avrdude -p atmega328p -c arduino -P /dev/ttyACM0 -b 115200 -U flash:w:mcu.hex:i
```

- p Το μοντέλο του μικροελεγκτή, το οποίο στην περίπτωση της υλοποίησης είναι ο ATmega328P.
- c Ο χρησιμοποιούμενος προγραμματιστής, ώστε να αναγνωρίζεται η συνδεσμολογία με το μικροελεγκτή και να εφαρμόζονται τα κατάλληλα σήματα ελέγχου.
- P Η θύρα του υπολογιστή στην οποία είναι συνδεδεμένος ο προγραμματιστής. Στην περίπτωση της υλοποίησης όπου χρησιμοποιείται Boot loader, η θύρα σύνδεσης είναι USB η οποία, στο σύστημα του υπολογιστή, εμφανίζεται ως /dev/ttyACM0.
- b Ο ρυθμός baud μετάδοσης δεδομένων μέσω της σειριακής. Σύμφωνα με τις προδιαγραφές του Arduino Uno, ο χρησιμοποιούμενος Boot loader λειτουργεί στα 115200Bd (Arduino, 2014a).
- U Ο τύπος της εργασίας που πρόκειται να εκτελεστεί. Το πρόγραμμα avrdude μπορεί να χρησιμοποιηθεί τόσο για την εγγραφή όσο και για την ανάγνωση των διαθέσιμων μνημών του εκάστοτε μικροελεγκτή (για παράδειγμα, Flash, EEPROM, fuse), εφόσον αυτό υποστηρίζεται/επιτρέπεται από τον μικροελεγκτή (Wunsch, 2013). Στο πλαίσιο της υλοποίησης, ενδιαφέρει μόνο η εγγραφή (w) της μνήμης Flash με τα δεδομένα του αρχείου HEX (μορφής intel).

### 2.1.2 Λοιπό λογισμικό ανάπτυξης

Πέραν των εργαλείων της συλλογής AVR GCC που περιγράφονται στην ενότητα Εργαλεία προγραμματισμού (σ. 5), χρησιμοποιούνται και ορισμένα άλλα. Όλα

πρόκειται για Ελεύθερο λογισμικό αδείας συμβατής είτε με την άδεια GNU GPL είτε την LGPL, με τα περισσότερα να υποστηρίζουν πολλαπλά λειτουργικά συστήματα πέραν των βασισμένων σε UNIX.

**Make** Εργαλείο αυτοματισμού της ενημέρωσης παρακολουθούμενων αρχείων, αναγνωρίζει, βάσει κανόνων που του δηλώνει ο χρήστης, ποια αρχεία έχουν τροποποιηθεί και ενεργοποιεί τις αντίστοιχες εντολές για την ενημέρωσή αυτών καθώς και πιθανών άλλων αρχείων εξαρτώμενων από αυτών που μόλις ενημερώθηκαν (Morse, McGrath και Frysinger, 2014).

Στο πλαίσιο της υλοποίησης, χρησιμοποιείται για την εκ νέου μεταγλώττιση μόνο εκείνων των αρχείων πηγαίου κώδικα που έχουν τροποποιηθεί, παράγοντας τα αντίστοιχα αρχεία αντικείμενου προγράμματος και την, εν συνεχείᾳ, σύνδεσή τους για την παραγωγής του αρχείου HEX. Επίσης, περιλαμβάνει τη μεταφορά του αρχείου στο μικροελεγκτή καθώς και τη σύνταξη της τεκμηρίωσης (βλ. επόμενο εργαλείο).

**LATEX** Σύστημα προετοιμασίας εγγράφων υψηλής τυπογραφικής αξίας, ορίζει ειδική σήμανση κειμένου που χρησιμοποιείται για το χαρακτηρισμό του κειμένου και μέσω ορισμένων εργαλείων, παράγεται το τελικό έγγραφο (Oetiker, 2011). Η παρουσίαση/εμφάνιση του εγγράφου (όπως περιιδρία, διατάξεις) καθορίζεται από, τροποποιήσιμους ένας ένα βαθμό, προκαθορισμένους εσωτερικούς κανόνες του συστήματος LATEX που έχουν δημιουργηθεί ώστε να ευνοείται η αναγνωσιμότητα του τελικού προϊόντος.

Το σύστημα LATEX και τα σχετικά εργαλεία χρησιμοποιούνται για τη σύνταξη της τεκμηρίωσης (του παρόντος εγγράφου). Μεταξύ άλλων, διευκολύνεται η εργασία με αναφορές, τόσο μεταξύ μερών του εγγράφου όσο και των βιβλιογραφικών αναφορών και η διατύπωση μαθηματικών παραστάσεων. Επίσης, καθώς ο επεξεργαστής κειμένου είναι υπεύθυνος για την εμφάνιση μόνο κειμένου, η απόχριση του είναι ταχύτερη και ο χειρισμός του πιο αποτελεσματικός.

Τέλος, επειδή τα αρχεία είναι κειμένου και όχι δυαδικά, οι αλλαγές που πραγματοποιούνται σε αυτά είναι δυνατό να παρακολουθούνται από το σύστημα παρακολούθησης αλλαγών σε επίπεδο γραμμής (βλ. επόμενο εργαλείο).

**Git** Σύστημα παρακολούθησης αλλαγών το οποίο χαρακτηρίζεται για την ταχύτητα τη μικρή επιβάρυνση σε πόρους συστήματος και την ευελιξία του (Git, 2014). Επιτρέπει, μεταξύ άλλων, την καταγραφή των αλλαγών που πραγματοποιούνται σε αρχεία της υλοποίησης (κατά κύριο λόγο, του πηγαίου κώδικα υλοποίησης και τεκμηρίωσης) και παρέχει τη δυνατότητα πραγματοποίησης

εύκολα αναστρέψιμων πειραματικών αλλαγών. Επιπλέον, παρέχει μία εύληπτη παρουσίαση πορείας των εργασιών που έχουν πραγματοποιηθεί.

**Doxxygen** Λογισμικό παραγωγής τεκμηρίωσης πηγαίου κώδικα μέσω ειδικών σημάνσεων που εισάγονται στα σχόλιά του, χυρίως απευθυνόμενο σε κώδικα γλώσσας C και C++ (Doxygen, 2014). Το σύνολο του πηγαίου κώδικα της υλοποίησης (για παράδειγμα, συναρτήσεις, δομές, μεταβλητές) τεκμηριώνεται σύμφωνα με τις οδηγίες του Doxygen ώστε να είναι δυνατή η δημιουργία ενός, ηλεκτρονικής μορφής, προγραμματιστικού εγχειριδίου των επιμέρους μονάδων (module) για την υποστήριξη της κατανόησης της λειτουργίας του καθενός και τη διευκόλυνση της ενσωμάτωσης τους σε πιθανές άλλες υλοποιήσεις.

**Qucs** Λογισμικό με γραφικό περιβάλλον (GUI) για τη δημιουργία και προσομοίωση κυκλωμάτων, με δυνατότητα απεικόνισης των αποτελεσμάτων των προσομοιώσεων σε διάφορες μορφές, όπως γραφήματα (Qucs, 2014). Χρησιμοποιείται για το σχεδιασμό του προτύπου της διασύνδεσης του μικροελεγκτή με όλα τα ολοκληρωμένα κυκλώματα και τα συμπληρωματικά στοιχεία της υλοποίησης. Επιλέγεται, χυρίως, λόγω της ευκολίας με την οποία δημιουργούνται προσαρμοσμένα κυκλώματα και επειδή υποστηρίζει την εξαγωγή των κυκλωμάτων σε διανυσματική μορφή που εύκολα ενσωματώνεται στο έγγραφο της τεκμηρίωσης, δίχως απώλειες στην ποιότητά τους.

**Inkscape** Λογισμικό σχεδίασης διανυσματικών γραφικών (Inkscape, 2014). Χρησιμοποιείται για τη δημιουργία όλων των επεξηγηματικών σχημάτων της τεκμηρίωσης για το λόγο ότι επιτρέπει γρήγορη σχεδίαση γραφικών υψηλής ευχρίνειας με δυνατότητα εύκολης ενσωμάτωσης στην τεκμηρίωση.

**Blender3D** Ολοκληρωμένη σουίτα δημιουργίας τρισδιάστατων γραφικών (Blender Foundation, 2014). Στο πλαίσιο της υλοποίησης, χρησιμοποιείται στο κομμάτι της κατασκευής της συσκευής, τόσο για το σχεδιασμό και εξέταση προτύπων όσο και για την εξαγωγή απεικονίσεων του εφαρμοσμένου προτύπου που περιλαμβάνονται, χυρίως, στο κεφάλαιο Κατασκευή (σ. 49).

**Gimp** Λογισμικό επεξεργασίας εικόνας (Gimp, 2014). Χρησιμοποιείται για τη βελτίωση των χαρακτηριστικών ορισμένων εικόνων (για παράδειγμα, αντίθεση, χορευμός), ώστε να παρέχεται καλύτερο οπτικό αποτέλεσμα.

## 2.2 Μικροελεγκτής

Σύμφωνα με τον Myklebust (1997, σ. 1), οι μικροελεγκτές AVR διαθέτουν μειωμένο σύνολο εντολών, δηλαδή είναι υπολογιστές RISC (Reduced Instruction Set Computer). Το χαρακτηριστικό αυτό απλοποιεί τα απαιτούμενα κυκλώματα ελέγχου και τους παρέχουν μικρότερους κύκλους για την εκτέλεση κάθε εντολής (Séquin και Patterson, 1982, σ. 1). Επιπροσθέτως, οι μικροελεγκτές AVR βασίζονται σε τροποποιημένη αρχιτεκτονική Harvard σύμφωνα με την οποία, και σε αντίθεση με την κατά Von Neumann αρχιτεκτονική, το πρόγραμμα και τα δεδομένα τοποθετούνται σε ανεξάρτητα φυσικά μέσα που χαρακτηρίζονται, μεταξύ άλλων, από ανεξάρτητους διαύλους πρόσβασης (Myklebust, 1997, σ. 1).

Άμεσα πλεονεκτήματα αυτού του σχεδιασμού είναι ότι καθίσταται δυνατή η ταυτόχρονη πρόσβαση στις μνήμες προγράμματος και δεδομένων στον ίδιο κύκλο (Atmel, 2013, σ. 8). Επιπλέον, επιτρέπεται η χρήση διαφορετικών τεχνολογιών για κάθε μνήμη. Για παράδειγμα, στην περίπτωση του χρησιμοποιούμενου μικροελεγκτή, ATmega328P, τα δεδομένα οργανώνονται σε μνήμη πλάτους των 8bit τεχνολογίας SRAM (Static RAM) με 2KiB συνολική χωρητικότητα, ενώ οι εντολές, σε μνήμη Flash των 32KiB με θέσεις των 16bit (Atmel, 2013, σσ. 8–9, 16, 18).

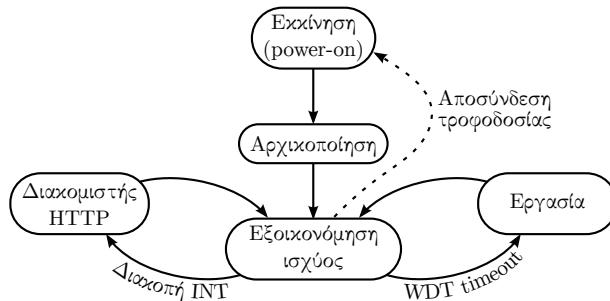
### 2.2.1 Μνήμη προγράμματος

Η μνήμη Flash του μικροελεγκτή χωρίζεται σε δύο περιοχές λογισμικού, την περιοχή του προγράμματος (Application section) και την περιοχή του λογισμικού Boot loader (Boot Loader Section – BLS) (Atmel, 2013, σ. 269). Η περιοχή προγράμματος φέρει τον κώδικα που εκτελεί ο μικροελεγκτής κατά την τυπική του λειτουργία· τον κώδικα της υλοποίησης. Στην περιοχή BLS εναποτίθεται λογισμικό το οποίο μπορεί να εγγράψει και να διαβάσει τη μνήμη Flash με δεδομένα που μεταφέρονται μέσω κάποιας διαλέσιμης διεπαφής του μικροελεγκτή (για παράδειγμα, USART ή SPI), που, τυπικά, χρησιμοποιείται για την εναπόθεση του νέου κώδικα του προγράμματος (Atmel, 2013, σσ. 269, 273).

Η εκτέλεση του Boot loader πραγματοποιείται είτε με την μεταπήδηση (εντολές JMP ή CALL) στην περιοχή BLS από την περιοχή προγράμματος, είτε μέσω μίας ρύθμισης του μικροελεγκτή που προκαλεί τη χρήση ως ρουτίνα εξυπηρέτησης της διακοπής επανεκκίνησης (Reset), την πρώτη διεύθυνση της περιοχής BLS αντί της πρώτης εντολής του προγράμματος, (Atmel, 2013, σ. 273). Από εκεί, το λογισμικό Boot loader είναι υπεύθυνο να αποφασίσει εάν απαιτείται εγγραφή νέου κώδικα στην περιοχή προγράμματος ή απευθείας μεταπήδηση πίσω σε αυτήν.

Το μέγεθος της περιοχής BLS είναι ρυθμιζόμενο και, στην περίπτωση του ATmega328, δύναται να καταλαμβάνει τις τελευταίες 256, 512, 1024 ή 2048 λέ-

Σχήμα 2.2.1: Καθήκοντα του μικροελεγκτή.



ξεις της μνήμης Flash (Atmel, 2013, σ. 282). Στην περίπτωση του Arduino Uno revision 3, το λογισμικό Boot loader καταλαμβάνει 512KiB (Arduino, 2014b).

## 2.2.2 Καθήκοντα μικροελεγκτή

Κατά την εκκίνηση της συσκευής, δηλαδή κατά τη σύνδεσή της με την παροχή τροφοδοσίας και, για λόγους που αναφέρονται στις ενότητες Πλατφόρμα Arduino (σ. 3) και Μνήμη προγράμματος (σ. 11), αφιερώνονται μερικά δευτερόλεπτα για την εκκίνηση πιθανού προγραμματισμού του μικροελεγκτή. Στην τυπική λειτουργία της συσκευής, αυτό γίνεται αντιληπτό ως μία μικρή καθυστέρηση κατόπιν της οποίας, εκτελέσται η αρχικοποίηση των διαφόρων υποσυστημάτων της υλοποίησης και, εν συνεχείᾳ, ο μικροελεγκτής μεταπίπτει σε κατάσταση χαμηλής κατανάλωσης ισχύος. Από αυτήν, ενεργοποιείται αυτόματα είτε για την εκκίνηση ενός νέου κύκλου μετρήσεων (βλ. Εργασία σ. 31), είτε για την εξυπηρέτηση κάποιου εισερχόμενου αιτήματος HTTP (βλ. Πόροι υλοποίησης σ. 133). Στο σχήμα 2.2.1 παρουσιάζεται ο κύκλος καθηκόντων του μικροελεγκτή.

Αναλυτικότερα, κατά το στάδιο της αρχικοποίησης, τίθεται η συχνότητα του ρολογιού του μικροελεγκτή, η οποία, για τις ανάγκες της υλοποίησης, μειώνεται από τα 16MHz στα 4MHz, και η κατεύθυνση των ακροδεκτών (δηλαδή ποιοι είναι εισόδου και ποιοι εξόδου). Επίσης, ρυθμίζεται το κύκλωμα WDT (Watch-Dog Timer), το οποίο είναι υπεύθυνο για την περιοδική αφύπνιση του μικροελεγκτή ώστε να ελέγχεται η ανάγκη εκκίνησης νέου κύκλου μετρήσεων.

Επιπλέον, ανακτώνται οι μεταβλητές ρυθμίσεις της συσκευής είτε από τις προκαθορισμένες (εργοστασιακές ρυθμίσεις), είτε από τις αποθηκευμένες τιμές της εφεδρικής μνήμης, εφόσον αυτές είναι έγκυρες (βλ. Εφεδρική μνήμη σ. 24). Οι ρυθμίσεις αυτές περιλαμβάνουν τη δικτύωση της συσκευής (διεύθυνση IP, μάσκα υποδικτύου, προεπιλεγμένη πύλη), το λειτουργικό εύρος του υποσυστήματος κίνησης (βλ. Συντεταγμένες και Λειτουργικό εύρος σ. 87), το χρονικό διάστημα μεταξύ

κύκλων εργασίας καθώς και το πλήθος μετρήσεων που πραγματοποιείται σε κάθε κύκλο (βλ. Εργασία σ. 31). Ο τρόπος ρύθμισης της συσκευής περιγράφεται στους Πόρους υλοποίησης (σ. 133).

Στην πορεία, οι ρυθμίσεις προωθούνται στις κατάλληλες μονάδες και εκτελούνται επιπρόσθετες προετοιμασίες, όπως η παλινόστηση της κεφαλής, δηλαδή η επαναφορά της στη θέση επιστροφής (συντεταγμένες  $[0, 0, Z_{\max}]$ ) (βλ. σ. 101) και η αρχικοποίηση του HTTP Socket (βλ. σ. 112).

### 2.2.3 Κατάσταση χαμηλής κατανάλωσης

Ο μικροελεγκτής διαθέτει διάφορες καταστάσεις νάρκης (sleep mode όπου η καθεμία απενεργοποιεί ορισμένα κυκλώματα για τη μείωση της κατανάλωσης. Για παράδειγμα, η πιο απλή, είναι η κατάσταση αδράνειας (idle) κατά την οποία απενεργοποιείται μόνο το ρολόι της CPU και της μνήμης Flash (δηλαδή, της μνήμης προγράμματος). Στο πλαίσιο της υλοποίησης, γίνεται προσπάθεια για την ύψιστη μείωση της κατανάλωσης κατά τα διαστήματα όπου ο μικροελεγκτής παραμένει άεργος.

Για το σκοπό αυτό, εφαρμόζεται η κατάσταση power-down κατά την οποία απενεργοποιούνται όλα τα ρολόγια του μικροελεγκτή ( $\text{clk}_{\text{CPU}}$ ,  $\text{clk}_{\text{FLASH}}$ ,  $\text{clk}_{\text{IO}}$ ,  $\text{clk}_{\text{ADC}}$ ,  $\text{clk}_{\text{ASY}}$ ) καθώς και οι ταλαντωτές του συστήματος και των Χρονομετρήσων/Απαριθμητών (Atmel, 2013, σ. 38). Σύμφωνα με την Atmel (2013, σ. 38), ο μικροελεγκτής είναι δυνατό να επανέλθει σε κανονική λειτουργία μόνο μέσω των ακροδεκτών INT1:0 με παρατεταμένο λογικό 0 (low-level interrupt) καθώς και μέσω των κυκλωμάτων TWI (Two-Wire Interface) και WDT (Watch-Dog Timer).

Στο πλαίσιο της υλοποίησης χρησιμοποιείται ο πρώτος και ο τελευταίος τρόπος· το κύκλωμα TWI αφυπνίζει το μικροελεγκτή όταν αυτός ρυθμίζεται με ρόλο slave στο δίσιλο TWI και κάποιο εξωτερικό κύκλωμα προσπαθεί να επικοινωνήσει μαζί του ενώ, στην υλοποίηση, χρησιμοποιείται μόνο ως master για την επικοινωνία με το ρολόι πραγματικού χρόνου (RTC) (βλ. Ωρα συστήματος σ. 30).

Για την αρρίβεια, ο ένας εκ των δύο ακροδεκτών INT1:0 συνδέεται με τον ακροδέκτη  $\overline{\text{INT}}$  του ολοκληρωμένου δικτύου, W5100, το οποίο τον θέτει και τον διατηρεί σε λογικό 0 έως ότου διευθετηθούν όλες οι ενδείξεις διακοπών που του έχουν ενεργοποιηθεί (βλ. Διασύνδεση με μικροελεγκτή σ. 107). Για τις ανάγκες της υλοποίησης, αυτό σημαίνει ότι έχει καταφύγασει εισερχόμενο αίτημα HTTP το οποίο διεκπεραιώνεται από το διακομιστή (βλ. Υποσύστημα διακομιστή HTTP σ. 119).

Ο χρονομετρητής WDT ρυθμίζεται ώστε να αφυπνίζει το μικροελεγκτή κάθε 8s (το μέγιστο διάστημα που υποστηρίζεται από τον παρόντα μικροελεγκτή) και

αποτελεί το έναυσμα για την εκκίνηση νέου κύκλου εργασιών (Εκκίνηση εργασίας σ. 35).

## 2.3 Η υλοποίηση συνοπτικά

### 2.3.1 Κατασκευή

Η κατασκευή της συσκευής γίνεται με χρήση ανοικτού (open hardware) συστήματος κατασκευής (construction framework), το OpenBuilds, για την παροχή διαφόρων διευκολύνσεων (ράγες κίνησης, γραμμική κίνηση).

Η συσκευή είναι εμπνευσμένη από τη διάταξη κινητού γεφυρώματος (moving gantry) εργαλειομηχανών CNC για την επίλυση των αναγκών της για γραμμική κίνηση στο χώρο, με εξαίρεση ότι αφαιρείται η τράπεζα. Η κίνηση αφορά την κεφαλή, ένα εξάρτημα που φέρει αισθητήρες, και μετακινείται σε επίπεδο πάνω από την επιφάνεια του παρακολουθούμενο υλικού και κατακόρυφα προς αυτό για την πραγματοποίηση μετρήσεων. Η ύπαρξη της κινητής κεφαλής επιτρέπει την αξιοποίηση ενός μικρού αριθμού αισθητήρων για την κάλυψη όλου του παρακολουθούμενου υλικού. (Βλ. Κατασκευή σ. 49.)

### 2.3.2 Κωδικοποίηση κίνησης

Η μετατόπιση της κεφαλής πραγματοποιείται από κινητήρες. Η κίνησή τους παρακολουθείται από κωδικοποιητή περιστροφικής κίνησης, ο οποίος παρέχει ανατροφοδότηση στο μικροελεγκτή για τα, αναφερόμενα ως, βήματα που πραγματοποιεί ο παρακολουθούμενος κινητήρας ώστε να αναγνωρίζει το μέγεθος της μετατόπισης. Τέσσερα τέτοια βήματα (ή παλμοί) αποτελούν μία πλήρη περιστροφή.

Ο κωδικοποιητής είναι αυτοσχέδιος και χρησιμοποιεί ανακλαστικό αισθητήρα υπερύθρων ακτίνων· μέρος της έντασης των ακτίνων που εκπέμπει ο πομπός που διαλέτει, καταφύγοντας στο δέκτη του, αφού πρώτα ανακλαστούν σε λωρίδες εναλλασσόμενης ανακλαστικότητας που κοσμούν την άτρακτο του κινητήρα. Παρέχεται ένας κωδικοποιητής ανά κινητήρα.

Η απόσταση του αισθητήρα από την άτρακτο καθώς και το πλάτος των λωρίδων επιλέγονται σύμφωνα με τις προδιαγραφές του αισθητήρα ώστε να ευνοείται η ικανότητά του να διαχρίνει τις εναλλαγές.

Ο κωδικοποιητής λειτουργείται με χαμηλή ένταση ρεύματος και μόνο κατά τα διαστήματα που κινείται ο αντίστοιχος κινητήρας για τη μείωση της συνολικής κατανάλωσης ισχύος της συσκευής και για την επιμήκυνση της διάρκειας ζωής του (πομπού του) αισθητήρα. (Βλ. Κωδικοποιητής κίνησης σ. 67.)

### 2.3.3 Υποσύστημα κίνησης

Η θέση της κεφαλής προσδιορίζεται από σύστημα συντεταγμένων τριών αξόνων X, Y και Z. Η ελάχιστη μετατόπιση ανά άξονα ορίζεται ως μία πλήρη περιστροφή της ατράκτου του κινητήρα, πρακτικά 4cm.

Το σήμα ελέγχου της κίνησης των κινητήρων παράγεται μέσω κυκλώματος Χρονομετρητή/Απαριθμητή (Timer/Counter) του μικροελεγκτή. Η λήξη της κίνησης αναλαμβάνεται από δεύτερο κύκλωμα Χρονομετρητή/Απαριθμητή ο οποίος καταμετρά το πλήθυσμα των βημάτων του κωδικοποιητή και τερματίζει την αναμετάδοση του σήματος κίνησης στον κινητήρα όταν ολοκληρώνεται το προκαθορισμένο πλήθυσμα βημάτων. Η χρήση του δεύτερου κυκλώματος Χρονομετρητή/Απαριθμητή επιτρέπει την παύση της μετατόπισης ακόμα και εάν η CPU του μικροελεγκτή είναι απασχολημένη.

Υλοποιείται παράλληλη κίνηση της κεφαλής στους άξονες X και Y η οποία βασίζεται στην παραδοχή ότι επιτυγχάνεται, και για τους δύο άξονες, η ίδια γωνιακή ταχύτητα για το χρονικό διάστημα κατά τον οποίο λειτουργούν παράλληλα. Πρακτικά, αυτό αποτελεί μία από τις προκλήσεις της υλοποίησης και χρήζει βελτίωσης, όπως αναφέρεται και παρακάτω (Βελτιώσεις σ. 143).

Για την κάλυψη κάθε ενδεχομένου, χρησιμοποιούνται ανασταλτικοί διακόπτες στις άκρες της συσκευής (όρια αξόνων). Όταν κάποιος εξ αυτών ενεργοποιηθεί, η κεφαλή επαναφέρεται σε μία συγκεκριμένη θέση (την αρχική ή θέση επιστροφής). Από εκεί, επιχειρείται μία ακόμη φορά η εκ νέου μετάβαση στην θέση που προηγουμένως κατέστη αδύνατη. Εφόσον ενεργοποιηθεί κάποιος διακόπτης ξανά, η μετάβαση ακυρώνεται. (Βλ. Υποσύστημα κίνησης σ. 85.)

### 2.3.4 Μετρήσεις

Η ύπαρξης της κεφαλής συνδέεται με την πραγματοποίηση μετρήσεων οι οποίες αποθηκεύονται στην εσωτερική μνήμη EEPROM του μικροελεγκτή. Οι, κατά μέγιστο, 90 πλέον πρόσφατες καταγεγραμμένες μετρήσεις, φέρουν την ημερομηνία και τη θέση στο επίπεδο X-Y όπου πραγματοποιήθηκε, καθώς και τη θερμοκρασία που σημειώθηκε. Η ημερομηνία/ώρα τηρείται από εξωτερικό ολοκληρωμένο ρολόι πραγματικού χρόνου (RTC) το οποίο τροποποιείται, καταλλήλως, από το χρήστη.

Η διαχείριση των εγγραφών γίνεται από τη μονάδα του ημερολογίου (log), η οποία είναι υπεύθυνη και για τη διατήρηση της εγκυρότητας των περιεχόμενων εγγραφών. Σε περίπτωση που πραγματοποιηθεί μέτρηση σε ημερομηνία προγενέστερη (σύμφωνα με την ώρα του RTC) σε σχέση με υπάρχουσες εγγραφές, το ημερολόγιο εγγράφει τη νέα μέτρηση αποβάλλοντας τις προϋπάρχουσες προγενέστερες (λογική διαγραφή και όχι φυσική για την μείωση φυλορών στην EEPROM).

Πίνακας 2.3.1: Εργοστασιακές ρυθμίσεις δικτύωσης της συσκευής.

Διεύθυνση IP :	192.168. 1.73
Προεπιλεγμένη πύλη :	192.168. 1. 1
Μάσκα υποδικτύου :	255.255.255. 0

Μετρήσεις πραγματοποιούνται αυτόματα από τη συσκευή ανά σταυρεά χρονικά διαστήματα καθορίζομενων από το χρήστη, σε ένα πλήθος τυχαία επιλεγμένων θέσεων. Το πλήθος των μετρήσεων καθορίζεται, επίσης, από το χρήστη. Μία σειρά μετρήσεων αναφέρεται ως εργασία και ζεκινά σε διαστήματα πολλαπλάσια των έξι λεπτών της ώρας. Ο μέγιστος χρόνος αδράνειας της συσκευής που υποστηρίζεται είναι, πέραν της εξ ολοκλήρου απενεργοποίησης των εργασιών, μία ημέρα. (Βλ. Λειτουργίες υποδομής σ. 17.)

### 2.3.5 Επικοινωνία με συσκευή

Η επικοινωνία της συσκευής με εξωτερικές οντότητες γίνεται μέσω πρωτοκόλλου HTTP· ο χρήστης αλληλεπιδρά μαζί της μέσω λογισμικού πλοήγησης (browser) ενώ τρίτα συστήματα, μέσω προγραμματιστικής διεπαφής (API). Για την ακρίβεια, η ίδια η διεπαφή χρήστη αποτελεί έναν καταναλωτή αυτής της προγραμματιστικής διεπαφής (API), και είναι υλοποιημένη με HTML5, CSS3 και Javascript (χωρίς επιπρόσθετες βιβλιοθήκες) σε βαθμό ώστε να παρέχεται συμβατότητα με τις περισσότερες εκδόσεις των δημοφιλέστερων λογισμικών πλοήγησης και για εκδόσεις Internet Explorer 8 και άνω.

Τα δεδομένα που είναι απαραίτητα για τη διεπαφή του χρήστη (ιστοσελίδα και αρχεία .css .js και εικόνων) εναποτίθενται σε εξωτερικό κύκλωμα μνήμης Flash 128KiB το οποίο χρησιμοποιείται μόνο για την ανάγνωσή τους και την επιστροφή τους μέσω του υποσυστήματος διακομιστή HTTP. Τα δεδομένα που ανταλλάσσονται μεταξύ τρίτων συστημάτων και συσκευής βρίσκονται σε αναπαράσταση JSON (πέραν των HTML, CSS, Javascript και PNG που προορίζονται για τη διεπαφή χρήστη).

Μέσω της διεπαφής της συσκευής δύναται η εμφάνιση των μετρήσεων που έχουν πραγματοποιηθεί καθώς και οι ρυθμίσεις της συμπεριφοράς της. Οι ρυθμίσεις διατηρούνται ακόμη και με την αποσύνδεση της συσκευής από την κεντρική τροφοδοσία. Η επαναφορά τους στις εργοστασιακές (προκαθορισμένες) τιμές γίνεται μέσω της προσωρινής απομάκρυνσης της μπαταρίας (coin cell) της συσκευής ενώ αυτή είναι αποσυνδεδεμένη από την κεντρική τροφοδοσία. Σημειώνεται ότι οι εργοστασιακές ρυθμίσεις απενεργοποιούν τις εργασίες, ενώ οι ρυθμίσεις δικτύου είναι αυτές που ορίζονται στον πίνακα 2.3.1. (Βλ. Διαδικτύωση σ. 103.)

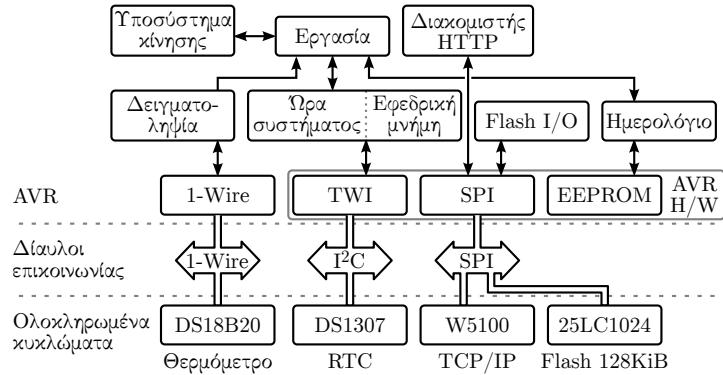
## Κεφάλαιο 3

# Λειτουργίες υποδομής

Το κεφάλαιο αναφέρεται σε ορισμένες βασικές μονάδες της υλοποίησης. Μία εξ αυτών είναι τα αρχεία της διεπαφής, τα οποία είναι υπεύθυνα για την ιστοσελίδα της συσκευής που παρέχει ένα γραφικό περιβάλλον επικοινωνίας μεταξύ χρήστη και συσκευής (βλ. Πόροι αρχείων σ. 139). Οι απαιτήσεις των αρχείων αυτών σε αποθηκευτικό χώρο είναι δύσκολο να ικανοποιηθούν είτε από τη μνήμη EEPROM είτε τη Flash του μικροελεγκτή, καθώς η πρώτη ανέρχεται μόλις στο 1KiB, ενώ η δεύτερη, στα 32KiB, η οποία χρησιμοποιείται, κατά κύριο λόγο, για την αποθήκευση του λογισμικού (προγράμματος) του μικροελεγκτή και θα ήταν δύσκολο να χρησιμοποιηθεί και για τα αρχεία της διεπαφής. Για το λόγο αυτό επιλέγεται ένα ολοκληρωμένο μνήμης Flash χωρητικότητας 1024bit (128KiB) για την αποθήκευση των αρχείων της διεπαφής (ιστοσελίδα, αρχείο css κ.ο.κ.). Τα δεδομένα της συγκεκριμένης μνήμης αρχικοποιούνται (δηλαδή, εγγράφονται) σε ένα στάδιο πριν τον τελικό προγραμματισμό της συσκευής και, κατά τη διάρκεια της κανονικής της λειτουργίας, η μνήμη χρησιμοποιείται μόνο για ανάγνωση (βλ. Αρχεία διεπαφής σ. 19).

Επιπλέον, ορίζεται μία εφεδρική μνήμη η οποία χρησιμεύει για την αποθήκευση των ρυθμίσεων της συσκευής, δηλαδή τιμών που τίθενται από το χρήστη και επηρεάζουν τη συμπεριφορά της συσκευής (βλ. σ. 24). Ο λόγος ύπαρξής της είναι η διατήρηση των ρυθμίσεων μεταξύ διαδοχικών επανεκκινήσεων της συσκευής. Χαρακτηρίζεται ως «εφεδρική» επειδή τα δεδομένα που εναποτίθενται σε αυτήν θεωρούνται έγκυρα όσο η εφεδρική τροφοδοσία της συσκευής είναι διαθέσιμη, ενώ επαναφέρονται στις προκαθορισμένες τιμές (τις εργοστασιακές ρυθμίσεις) εφόσον και αυτή (επιπροσθέτως της κύριας τροφοδοσίας) απενεργοποιηθεί προσωρινά. Η εφεδρική τροφοδοσία πρόκειται για μία συστοιχία συσσωρευτών (μπαταρία) που επιτρέπει στο ρολόι πραγματικού χρόνου (RTC) να λειτουργεί κατά τα διαστήματα που η συσκευή είναι αποσυνδεδεμένη από την κύρια τροφοδοσία (δηλαδή,

Σχήμα 3.0.1: Σχέσεις μεταξύ των βασικών μονάδων της υλοποίησης.



όταν η συσκευή είναι απενεργοποιημένη).

Στη συνέχεια αναλύεται το λογισμικό που είναι υπεύθυνο για τη διαχείριση των μετρήσεων, το Ημερολόγιο (25). Το Ημερολόγιο αναλαμβάνει την αποθήκευση και ανάκτηση εγγραφών οι οποίες περιέχουν, υποχρεωτικά, ημερομηνία/ώρα και ένα πλήθος επιπρόσθετων Byte. Οι εγγραφές επιστρέφονται σε φθίνουσα σειρά βάσει της ημερομηνίας/ώρας κάθε εγγραφής ενώ η ανάκτησή τους υλοποιείται με τρόπο που διευκολύνει τη σελιδοποίηση των εγγραφών στο πλαίσιο αναζητήσεων (βλ. Πόρος /measurement σ. 137).

Ο αποθηκευτικός χώρος των εγγραφών του Ημερολογίου έχει επιλεγεί να είναι η εσωτερική μνήμη EEPROM του μικροελεγκτή (1 KiB). Πρακτικά, αυτό σημαίνει ότι το Ημερολόγιο είναι ρυθμισμένο να διατηρεί τις 90 πλέον πρόσφατες μετρήσεις, αφήνοντας μερικά Byte (34) για ενδεχόμενα άλλα δεδομένα.

Κατόπιν, ακολουθεί περιγραφή της διευθέτησης και χειρισμού του ρολογιού πραγματικού χρόνου (RTC – Real-Time Clock), ένα ολοκληρωμένο που παρέχει στη συσκευή τη δυνατότητα αναγνώρισης της τρέχουσας ημερομηνίας/ώρας (βλ. Ωρα συστήματος σ. 30). Στο παρόν στάδιο της υλοποίησης, το ολοκληρωμένο χρησιμοποιείται για την αναγνώριση του χρόνου που έχει παρέλθει από την τελευταία μέτρηση του Ημερολογίου ώστε να αποφασίζεται εάν πρέπει να εκκινηθεί νέος κύκλος μετρήσεων. Ο σχετικός έλεγχος πραγματοποιείται κάθε 8s από ξεχωριστή μονάδα, όταν ο μικροελεγκτής αφυπνίζεται από εσωτερικό κύκλωμά του, το Watch-Dog Timer (WDT).

Η υπεύθυνη μονάδα για την εκτέλεση των μετρήσεων – όπου στο πλαίσιο της υλοποίησης περιλαμβάνουν μόνο τη θερμοκρασία – είτε με αυτόματο τρόπο είτε μη περιγράφεται στην Εργασία (σ. 31). Στο πλαίσιο της, παρέχεται, επιπρόσθετως, η δυνατότητα εκτίμησης του χρόνου ολοκλήρωσης της τρέχουσας εργασίας (είτε αυτή αναφέρεται σε μεμονωμένη ή αλληλουχία μετρήσεων, είτε σε απλή μετακίνηση

της κεφαλής) το οποίο αξιοποιείται κατά την απόχριση σε ορισμένα αιτήματα (βλ. Πόροι υλοποίησης σ. 133).

Τέλος, αναλύονται οι δίαιυλοι και το λογισμικό οδήγησης που χρησιμοποιούνται για την επικοινωνία μεταξύ του μικροελεγκτή και των διαφόρων εξωτερικών ολοκληρωμένων. Συνολικά, γίνεται χρήση των διαύλων 1-Wire, I<sup>2</sup>C και SPI (βλ. Δίαιυλοι επικοινωνίας ολοκληρωμένων σ. 41).

Οι σχέσεις μεταξύ των διαφόρων μονάδων που αναλύονται σε αυτό το κεφάλαιο, εμφανίζονται στο σχήμα 3.0.1, με εξαίρεση του Υποσυστήματος κίνησης και της Διαδικτύωσης τα οποία διαθέτουν ξεχωριστά κεφάλαια (βλ. σ. 85 και σ. 103, αντίστοιχα).

## 3.1 Μνήμες αποθήκευσης

### 3.1.1 Αρχεία διεπαφής

Κρίνεται σκόπιμη η ύπαρξη ενός αφιερωμένου αποθηκευτικού μέσου για την εναπόθεση των αρχείων της διεπαφής της συσκευής (ιστοσελίδα, κώδικας javascript κ.ο.κ. – βλ. Πόροι αρχείων σ. 139) καθώς οι, εσωτερικά, διαθέσιμες μνήμες του μικροελεγκτή – 32KiB μνήμη προγράμματος και 1KiB μνήμη EEPROM – θεωρείται ότι αδυνατούν να τα υποστηρίζουν ή θα έθεταν αισθητούς περιορισμούς στο μέγεθός τους και, συνεπώς, στη λειτουργικότητα της διεπαφής.

Για το σκοπό αυτό, επιλέγεται το ολοκληρωμένο 25LC1024 το οποίο διαθέτει 128KiB μνήμη EEPROM, χωρισμένη σε τέσσερις τομείς των 32KiB με 128 σελίδες ο καθένας, με δυνατότητα εγγραφής σε επίπεδο Byte ή σελίδας και διασύνδεση μέσω διαύλου SPI (Microchip, 2003, σ. 1). Για λεπτομέρειες σχετικά με το δίαιυλο SPI, βλ. σ. 47. Κρίνεται ότι η χωρητικότητά του είναι υπεραρκετή για τις τρέχουσες, ή άμεσα μελλοντικές, ανάγκες της υλοποίησης. Στο παρόν στάδιο της υλοποίησης, η αποθήκευση των αρχείων στη μνήμη γίνεται σε αυθαίρετα επιλεγμένες διευθύνσεις μνήμης, οι οποίες δηλώνονται κατά τη σύνταξη του προγράμματος, ενώ τα αρχεία είναι αδύνατο να μετοβληθούν μέσα από την υλοποίηση.

Η πρόσβαση της μνήμης του ολοκληρωμένου για την ανάγνωση και εγγραφή των κειμών της γίνεται μέσω ορισμένων εντολών. Το σύνολο των εντολών μπορεί να χωριστεί σε τρεις κατηγορίες: εντολές ανάγνωσης, εγγραφής και ειδικής χρήσης, καθεμία από τις οποίες μελετάται ξεχωριστά. Στη γενική τους μορφή, δύες οι εντολές περιγράφονται από τους ακόλουθους κανόνες:

```
command = instruction_code [ load ]
load = memory_cell | octet
memory_cell = address *octet
```

Οι κωδικοί εντολών έχουν μήκος 8bit ενώ οι διευθύνσεις, 24bit εκ των οποίων τα πρώτα 7 αγνοούνται από το ολοκληρωμένο καθώς τα υπολειπόμενα 17 είναι ικανά για την αναφορά στις 128KiB διαυθέσιμες θέσεις (Microchip, 2003, σσ. 6–7). Σαφώς, τα πραγματικά μέρη που συνθέτουν την εκάστοτε εντολή εξαρτώνται από τον κωδικό και, συνεπώς, τη λειτουργία της. Για παράδειγμα, η διεύθυνση κελιού μνήμης (address) έχει νόημα για τις εντολές ανάγνωσης, εγγραφής και απαλοιφής μέρους της μνήμης, ενώ η απαλοιφή όλων των δεδομένων της μνήμης (Chip Erase) απαιτεί μόνο τον κωδικό της (instruction\_code).

Ωστόσο, όλες οι εντολές προϋποθέτουν ότι το 25LC1024 βρίσκεται σε κατάσταση ετοιμότητας (δηλαδή, όχι σε κατάσταση χαμηλής κατανάλωσης ισχύος), ενώ ενδέχεται να ισχύουν επιπρόσθετοι περιορισμοί, ανάλογα με την εντολή (Microchip, 2003, σ. 17).

### Ανάγνωση

Οι εντολές ανάγνωσης επιτρέπουν την ανάκτηση δεδομένων από τα κελιά της μνήμης καθώς και την τιμή του καταχωρητή κατάστασης SR (Status Register) – του μοναδικού καταχωρητή του ολοκληρωμένου – ο οποίος παρέχει στοιχεία για την πορεία ενδεχόμενων εγγραφών και έλεγχο γύρω από την ενεργοποίηση της εγγραφής (περισσότερα βλ. Εγγραφή και προστασία σ. 21).

Σύμφωνα με τη Microchip (2003, σ. 6), το ολοκληρωμένο αγνοεί τις απόπειρες ανάγνωσης μνήμης όταν βρίσκεται σε εξέλιξη εγγραφή δεδομένων, δηλαδή όσο η ένδειξη WIP (Write-In-Progress) του καταχωρητή κατάστασης βρίσκεται σε λογικό 1.

Ως εντολές ανάγνωσης μπορούν να χαρακτηριστούν οι ακόλουθες:

**READ [0x03]** Επιστρέφει το Byte της μνήμης που βρίσκεται στην προσδιοριζόμενη διεύθυνση και συνεχίζει με κάθε επόμενο για διαδοχικούς παλμούς της ίδιας εντολής, αυτομάτως μεταβαίνοντας στη διεύθυνση 0 ως το επόμενο Byte της τελευταίας διεύθυνσης (Microchip, 2003, σσ. 6–7). Προφανώς, η σύνταξη της εντολής απαιτεί διεύθυνση και τουλάχιστον ένα Byte δεδομένων.

**RDSR [0x05]** (ReaD Status Register) Επιστρέφει την τιμή του καταχωρητή κατάστασης του ολοκληρωμένου και απαιτεί, επιπρόσθετας του κωδικού εντολής, ένα Byte (Microchip, 2003, σ. 10).

**RDID [0xAB]** (ReaD ID) Επιστρέφει το αναγνωριστικό της συσκευής και την αφυπνίζει από την κατάσταση χαμηλής κατανάλωσης ισχύος, εάν έχει τεθεί σε αυτήν. Η εντολή απαιτεί την αποστολή 24bit διεύθυνσης (τα οποία χρησιμοποιούνται για την προετοιμασία του ολοκληρωμένου, χωρίς να έχει

Πίνακας 3.1.1

BP1	BP0	Προστατευμένη περιοχή
0	0	Καμία
0	1	3 <sup>ος</sup> τομέας (0x18000–0x1FFFF)
1	0	2 <sup>ος</sup> και 3 <sup>ος</sup> τομέας (0x10000–0x1FFFF)
1	1	Όλοι οι τομείς (0x00000–0x1FFFF)

Βασισμένο TABLE 2-3: ARRAY PROTECTION. Στο Microchip Technology. 25AA1024/25LC1024. 1 Mbit SPI<sup>TM</sup> Bus Serial Flash. 28 Ιούλ. 2003. 28 σσ. url: <http://ww1.microchip.com/downloads/en/DeviceDoc/22064D.pdf> (επίσκεψη 05/08/2014), σ. 11

ιδιαίτερη σημασία η τιμή τους), και ένα Byte για την επιστροφή του αναγνωριστικού (ή «υπογραφής») του (Microchip, 2003, σ. 17).

Για την εντολή μετάπτωσης σε κατάσταση χαμηλής κατανάλωσης ισχύος, βλ. Ειδικές εντολές (σ. 24).

### Εγγραφή και προστασία

Οι εγγραφή νοείται τόσο η αποστολή δεδομένων (Byte) από το μικροελεγκτή με προορισμό τον καταχωρητή κατάστασης SR του ολοκληρωμένου ή συγκεκριμένες θέσεις της μνήμης EEPROM, είτε η απαλοιφή των δεδομένων από τα κελιά της μνήμης και την επαναφορά τους στην προκαθορισμένη τιμή 0xFF (erase) (Microchip, 2003, σσ. 6-7).

Οστόσο, για την αποφυγή μη εσκεμμένων (τυχαίων) εγγραφών στις θέσεις μνήμης με αποτέλεσμα την αλλοίωση των δεδομένων, το ολοκληρωμένο απαιτεί τη δήλωση της πρόθεσης για εγγραφή. Επιπλέον, παρέχονται δύο επιπρόσθετα επίπεδα προστασίας ώστε να μειώνεται περαιτέρω η πιθανότητα ακούσιας τροποποίησης των δεδομένων από αστοχίες υλικού και εξωτερικών παρεμβολών.

**Μανδαλωτής WEL** Αρχικά, προκειμένου να εκτελεστεί οποιαδήποτε εντολή εγγραφής, απαιτείται να προηγηθεί η αποστολή της εντολής WREN (WRite ENable) ώστε να ενεργοποιηθεί ο μανδαλωτής WEL (Write-Enable Latch) (Microchip, 2003, σ. 6). Κατόπιν, απενεργοποιείται η γραμμή  $\overline{CS}$  (τίθεται σε λογικό 1) και ενεργοποιείται ξανά (λογικό 0) για την αποστολή της εντολής εγγραφής. Ο μανδαλωτής WEL απενεργοποιείται είτε αυτόματα με την ολοκλήρωση της εγγραφής ή κατά την αρχική τροφοδοσία του ολοκληρωμένου με τάση (power-on) είτε μέσω της, ειδικής για αυτόν τον σκοπό, εντολής WRDI (WRite DIable) (Microchip, 2003, σ. 9). Αυτό το επίπεδο προστασίας είναι το μοναδικό που εφαρμόζεται υποχρεωτικά.

Πίνακας 3.1.2: Επίπεδα προστασίας από εγγραφή.

WEL (SR bit 1)	WPEN (SR bit 7)	WP (pin 3)	Περιοχή BP1:0	Τυπόλοιπη μνήμη	Καταχωρητής SR
0	X	X	ναι	ναι	ναι
1	0	X	ναι	όχι	όχι
1	1	0 (low)	ναι	όχι	ναι
1	1	1 (high)	ναι	όχι	όχι

X : αδιάφορο

ναι : προστατευμένο από εγγραφή

όχι : εγγράψιμο

Βασισμένο TABLE 2-4: WRITE-PROTECT FUNCTIONALITY MATRIX. Στο Microchip Technology. 25AA1024/25LC1024. 1 Mbit SPI<sup>TM</sup> Bus Serial Flash. 28 Ιούλ. 2003. 28 σσ. url: <http://ww1.microchip.com/downloads/en/DeviceDoc/22064D.pdf> (επίσκεψη 05/08/2014), σ. 12

**Προστατευμένες περιοχές** Ως ένα επιπρόσθετο επίπεδο ασφαλείας είναι δυνατό να οριστεί, μέσω των bit BP1:0 του καταχωρητή SR, ποιος συνδυασμός τομέων θα είναι «προστατευμένος από εγγραφή», με την έννοια ότι η εγγραφή στις θέσεις μνήμης αυτών των τομέων απενεργοποιείται, εκτός και εάν προηγηθεί τροποποίηση αυτής της ρύθμισης σε προγενέστερο χρόνο της εντολής εγγραφής (Microchip, 2003, σσ. 10,12). Οι δυνατοί συνδυασμοί παρουσιάζονται στον πίνακα 3.1.1.

**Προστασία καταχωρητή SR** Το τελευταίο στάδιο προστασίας αφορά τον καταχωρητή SR και συμπεριλαμβάνει το χειρισμό του ακροδέκτη WP του ολοκληρωμένου επιπρόσθετως του μανδαλωτή WEL προκειμένου να επιτραπεί η τροποποίηση των μη πτητικών bit του καταχωρητή SR (δηλαδή των bit WPEN και BP1:0) (Microchip, 2003, σσ. 10–12). Η ενεργοποίηση αυτού του επιπέδου προστασίας πραγματοποιείται από το bit WPEN (Write-Protect ENable) του καταχωρητή SR. Εφόσον η δυνατότητα εγγραφής του καταχωρητή επαναφερθεί, ακολουθεί τροποποίηση των bit BP1:0 ώστε να επιτραπεί πρόσβαση στις προστατευμένες περιοχές.

Τα τρία επίπεδα προστασίας από εγγραφή παρουσιάζονται στον πίνακα 3.1.2 ενώ οι εντολές που υπάγονται στην κατηγορία εγγραφής αναφέρονται παρακάτω. Όλες οι εντολές αυτές υπόκεινται στην προστασία εγγραφής και αγνοούνται εάν, τη στιγμή υποβολής τους, ο μανδαλωτής WEL είναι απενεργοποιημένος ή εάν αναφέρονται σε κελί μνήμης που ανήκει σε προστατευμένη περιοχή (βλ. πίνακα 3.1.1). Για την ακρίβεια, ακόμα και οι εντολές SE και CE που εφαρμόζονται σε πολλαπλούς τομείς, αγνοούνται ολοκληρωτικά σε περίπτωση που αναφέρονται έστω και σε έναν προστατευμένο τομέα (Microchip, 2003, σσ. 14–15).

**WREN [0x06]** (WRite ENable) Όχι ακριβώς εντολή εγγραφής αλλά άμεσα σχε-

τιζόμενη. Όπως έχει αναφερθεί, προκειμένου να πραγματοποιηθεί εγγραφή στη μνήμη ή στον καταχωρητή κατάστασης, πρέπει πρώτα να ενεργοποιηθεί ο μανδαλωτής WEL (Write-Enable Latch) (Microchip, 2003, σ. 6). Η εντολή WREN κάνει ακριβώς αυτό για την επόμενη εντολή εγγραφής που αποστέλλεται στο ολοκληρωμένο, ενώ ταυτόχρονα θέτει την ένδειξη WEL (Write-Enable Latch) του καταχωρητή κατάστασης (Microchip, 2003, σ. 10). Η εντολή WREN αποτελείται μόνο από τον κωδικό της.

**WRDI [0x04]** (WRite DIable) Απενεργοποιεί χειροκίνητα το μανδαλωτή WEL χωρίς την εκτέλεση κάποιας εντολής εγγραφής (Microchip, 2003, σ. 9). Και αυτή η εντολή αποτελείται μόνο από τον κωδικό της.

**WRITE [0x02]** Επιτρέπει την αποστολή ενός ή περισσοτέρων Byte για εγγραφή στη μνήμη, υποχρεωτικά εντός μίας συγκεκριμένης σελίδας, ξεκινώντας από την προσδιοριζόμενη διεύθυνση εκείνης της σελίδας (Microchip, 2003, σσ. 6,8). Τα αποστελλόμενα Byte εναποτίθενται σε προσωρινό χώρο αποθήκευσης του ολοκληρωμένου (μέγιστης χωρητικότητας, το μέγεθος σελίδας – 256Byte), ενώ η πραγματική εγγραφή ξεκινάει και πραγματοποιείται ασύγχρονα κατόπιν απενεργοποίησης της γραμμής CS (Microchip, 2003, σ. 6).

Η ολοκλήρωση της εγγραφής σηματοδοτείται μέσω της ένδειξης WIP (Write-In-Progress) του καταχωρητή κατάστασης, ενώ κατά τη διάρκεια της εγγραφής, η μνήμη είναι αδύνατο να προσπελαστεί για πάσης φύσεως λειτουργία (Microchip, 2003, σ. 6).

**WRSR [0x01]** (WRite Status Register) Επιτρέπει την αλλαγή των μη πτητικών bit του καταχωρητή (δηλαδή των bit WPEN και BP1:0) που ελέγχουν τις προστατευμένες περιοχές της μνήμης (Microchip, 2003, σσ. 10–11). Ο κωδικός εντολής ακολουθείται από ένα Byte που περιέχει τις νέες ρυθμίσεις.

**PE [0x42]** (Page Erase) Δέχεται μία διεύθυνση και θέτει σε 0xFF όλα τα κελιάς της σελίδας στην οποία αυτή ανήκει (Microchip, 2003, σ. 13).

**SE [0xD8]** (Sector Erase) Αντίστοιχη της PE με τη διαφορά ότι εφαρμόζεται στις θέσεις μνήμης ολόκληρου του τομέα που περιέχει την αναφερόμενη διεύθυνση (Microchip, 2003, σ. 14).

**CE [0xC7]** (Chip Erase) Αντίστοιχη των PE και SE μόνο που εφαρμόζεται σε ολόκληρη τη μνήμη (Microchip, 2003, σ. 15).

Για την υλοποίηση κρίνεται αρκετή η χρήση του πρώτου (και υποχρεωτικού) επιπέδου προστασίας (μανδαλωτής WEL) κυρίως επειδή εγγραφή στη μνήμη πραγ-

ματοποιείται μόνο κατά την αρχική ρύθμιση του συστήματος και όχι κατά την τυπική του λειτουργία. Για την ακρίβεια, ο κώδικας που είναι υπεύθυνος για την εγγραφή των αρχείων της διεπαφής στη μνήμη χρησιμοποιείται μόνο προσωρινά και εξαιρείται από τον τελικό κώδικα που εγγράφεται στη μνήμη προγράμματος του μικροελεγκτή.

### Ειδικές εντολές

Σε αυτήν την κατηγορία συγκαταλέγονται οι εντολές που ελέγχουν πρόσθετες λειτουργίες του ολοκληρωμένου, ουσιαστικά, την κατάσταση χαμηλής κατανάλωσης ισχύος (Microchip, 2003, σ. 16).

**RDID [0xAB]** (ReaD ID) Επιστρέφει το αναγνωριστικό της συσκευής και την αφυπνίζει από την κατάσταση χαμηλής κατανάλωσης ισχύος (Microchip, 2003, σ. 17). Παρότι έχει συμπεριληφθεί στην κατηγορία εντολών ανάγνωσης επειδή πραγματοποιεί ανάγνωση του αναγνωριστικού του ολοκληρωμένου (βλ. σ. 21), αναφέρεται και σε αυτήν την κατηγορία για λόγους πληρότητας.

**DPD [0xB9]** (Deep Power-Down) Θέτει το ολοκληρωμένο σε κατάσταση χαμηλής κατανάλωσης ισχύος κατά την οποία όλες οι εντολές αγνοούνται, με εξαίρεση της RDID που χρησιμοποιείται για την αφύπνισή του (Microchip, 2003, σ. 16)

### 3.1.2 Εφεδρική μνήμη

Η συσκευή διαθέτει ένα πλήθος ρυθμίσεων που είναι δυνατό να τροποποιηθούν από το χρήστη ώστε να προσαρμόζεται η συμπεριφορά της σε διαφορετικές απαιτήσεις, όπως, για παράδειγμα, το χρονικό διάστημα μεταξύ μετρήσεων, το λειτουργικό εύρος (δηλαδή, οι διαστάσεις του παραχολουθούμενου χώρου) και οι δικτυακές ρυθμίσεις της συσκευής. Περισσότερα σχετικά με τις υποστηριζόμενες ρυθμίσεις περιγράφονται στην ενότητα Πόρος /configuration (σ. 134).

Όπως είναι αναμενόμενο, οι ρυθμίσεις πρέπει να διατηρούνται μεταξύ διαδοχικών επανεκκινήσεων της συσκευής, είτε αυτές οφείλονται σε προγραμματισμένες εργασίες συντήρησης είτε σε απρόσμενα περιστατικά (παραδείγματα, μετακίνηση σε νέο χώρο ή διακοπή ρεύματος, αντίστοιχα). Αυτομάτως, η χρήση μόνο της κύριας μνήμης του μικροελεγκτή για την τήρησή τους χρίνεται ακατάλληλη, καθώς πρόκειται για πτητική μνήμη (για την ακρίβεια, Static RAM) με αποτέλεσμα, σε κάθε επανεκκίνηση της συσκευής να ανακτώνται οι προκαθορισμένες τιμές του προγράμματος και όχι οι δηλωμένες από το χρήστη.

Επομένως, απαιτείται μία επιπρόσθετη, πιο μόνιμη μορφή αποθήκευσης των ρυθμίσεων, με άμεσους υποψηφίους, την εσωτερική μνήμη EEPROM 1KiB του

μικροελεγκτή ή την εξωτερική μνήμη Flash (ολοκληρωμένο 25LC1024, βλ. σ. 19). Ανεξαρτήτως της επιλογής, όταν πρέπει να δίνεται στο χρήστη η δυνατότητα επαναφοράς στις προκαθορισμένες (ή εργοστασιακές) ρυθμίσεις της συσκευής για την αντιμετώπιση περιπτώσεων αδυναμίας τροποποίησής τους μέσω της διεπαφής. Αντιπροσωπευτικό τέτοιο παράδειγμα αποτελεί η απώλεια της διεύθυνσης IP που έχει αποδοθεί στη συσκευή. Δεδομένου ότι στο τρέχον στάδιο της υλοποίησης υποστηρίζεται μόνο χειροκίνητη απόδοση διεύθυνσης, η διεπαφή είναι αδύνατο να προσπελαστεί εφόσον είναι άγνωστη η διεύθυνση IP της.

Η λύση παρέχεται από το RTC (Real-Time Clock) που χρησιμοποιείται για την τήρηση της τρέχουσας ημερομηνίας και ώρας. Όπως περιγράφεται στη σχετική ενότητα Ήρα συστήματος (σ. 30), το RTC τροφοδοτείται από την κεντρική γραμμή τροφοδοσίας της συσκευής και, σε περίπτωση πτώσης της σε χαμηλότερα επίπεδα από την εφεδρική τάση του RTC, μεταπίπτει αυτομάτως στη χρήση της δεύτερης. Εάν για κάποιο λόγο και η εφεδρική τροφοδοσία διακοπεί, το RTC παύει να λειτουργεί και παραμένει απενεργοποιημένο ακόμη και κατόπιν επαναφοράς της τροφοδοσίας.

Το ολοκληρωμένο διαθέτει σχετική ένδειξη λειτουργίας το οποίο μπορεί να ελεγχθεί προκειμένου να αποφασιστεί η ανάκτηση είτε των αποθηκευμένων ρυθμίσεων του χρήστη είτε οι εργοστασιακές. Εφόσον στο πλαίσιο της υλοποίησης η εφεδρική τροφοδοσία παρέχεται μέσω συστοιχίας συσσωρευτών, η επαναφορά στις εργοστασιακές ρυθμίσεις μπορεί, πολύ εύκολα, να πραγματοποιηθεί με την προσωρινή απομάκρυνση της μπαταρίας ενώ η συσκευή είναι απενεργοποιημένη.

Οστόσο, σημειώνεται ότι, στην περίπτωση της υλοποίησης, για την αποθήκευση των ρυθμίσεων χρησιμοποιείται η, γενικού σκοπού, μνήμη του RTC αντί κάποιας εκ των προαναφερθείσων μνημών EEPROM. Ο μόνος άμεσος περιορισμός είναι η χωρητικότητα της μνήμης του RTC – 56Byte – που για την υλοποίηση, ο χώρος αυτός είναι αρκετός.

Οστόσο, στην περίπτωση της υλοποίησης, και δεδομένου ότι διατίθεται σχετική υποδομή, χρησιμοποιείται η, γενικού σκοπού, μνήμη του RTC (Real-Time Clock) των 56Byte (βλ. Μνήμη RAM σ. 31). Ένας βασικός περιορισμός είναι το συνολικά διαθέσιμο μέγεθος. Τα 56Byte που διαθέτει το RTC είναι αρκετά για τις ανάγκες της υλοποίησης. Σε διαφορετική

## 3.2 Ημερολόγιο

Μία αναμενόμενη απαίτηση της υλοποίησης είναι η διατήρηση των μετρήσεων για την μετέπειτα ανάκτηση και επεξεργασία τους με σκοπό την εξαγωγή συμπερασμάτων ή τη λήψη αποφάσεων για την κατάσταση του μέσου. Το Ημερολόγιο

Σχήμα 3.2.1: Δομή μίας εγγραφής Ημερολογίου.

ημερομηνία							στοιχεία μέτρησης				
YY	MM	DD	HH	mm	ss	X	Y	T	RH	pH	

μετρήσεων αναλαμβάνει τη διαχείριση εγγραφών που παρέχουν αυτήν την πληροφορία. Ωστόσο το περιεχόμενο και η εγκυρότητα κάθε εγγραφής επαφίεται σε τρίτα υποσυστήματα που υποβάλλουν τις εκάστοτε μετρήσεις.

Η τήρηση του ημερολογίου έχει επιλεγεί να γίνεται στην εσωτερική μνήμη EEPROM του μικροελεγκτή, χωρητικότητας 1KiB.

### 3.2.1 Δομή εγγραφής

Σημείο κεντρικού ενδιαφέροντος του Ημερολογίου είναι οι εγγραφές. Καθότι ανήκει σε ημερολόγιο, κάθε εγγραφή είναι άρρηκτα συνδεδεμένη με μία ημερομηνία/ώρα (εφεξής αναφερόμενη απλά ως «ημερομηνία»), που προσδιορίζει τη χρονική τοποθέτησή της και χρησιμεύει ως χριτήριο αναζήτησης – το μοναδικό που υποστηρίζεται από το Ημερολόγιο σε αυτήν την υλοποίηση. Εκτός από την ημερομηνία, τηρούνται και τα στοιχεία της μέτρησης, όπως οι συντεταγμένες όπου έγινε η δειγματοληψία και, σαφώς, οι ενδείξεις των αισθητήριων οργάνων. Το σχήμα 3.2.1 παρουσιάζει τη δομή κάθε εγγραφής.

Είναι φανερό ότι η ημερομηνία καταλαμβάνει ένα πολύ μεγάλο μέρος της εγγραφής και ο λόγος είναι ότι αναπαριστάται σε BCD. Ο συμβολισμός BCD (Binary-Coded Decimal notation – Δυαδικά Κωδικοποιημένος Δεκαδικός συμβολισμός), όπως χρησιμοποιείται στην προκειμένη, απαιτεί ένα Byte για την αναπαράσταση οποιουδήποτε αριθμού του δεκαδικού συστήματος αρίθμησης από 0 μέχρι 99. εύρος ικανό να καλύψει τις ανάγκες του Ημερολογίου.

Κύριος λόγος για την επιλογή του συμβολισμού BCD για την ημερομηνία είναι η αμεσότητα μετατροπής της από αριθμό σε κείμενο, και αντιστρόφως. Πρακτικά, αυτό σημαίνει ότι είναι ταχύτερη η μετατροπή της κάθε εγγραφής και, κατ' επέκταση μίας πληθώρας εγγραφών, σε κείμενο που προορίζεται για εμφάνιση σε χρήστη.

Το μειονέκτημα είναι ότι δεσμεύεται πολύ περισσότερος χώρος από ότι ουσιαστικά αξιοποιείται. Για παράδειγμα, το ίδιο εύρος ημερομηνιών θα μπορούσε να καλυφθεί από έναν αριθμό 32-bit που διατηρεί το πλήθυσμα δευτερολέπτων που έχουν παρέλθει από την ημερομηνία αναφοράς (την παλαιότερη υποστηριζόμενη ημερομηνία). Ωστόσο, αυτή η προσέγγιση θα απαιτούσε την αναγωγή των συντιστωσών της ημερομηνίας (έτος, μήνας, μέρα κ.ο.κ.) κάθε φορά που χρειαζόταν μία

εξ αυτών.

### Δομή ημερομηνίας

Οι συνιστώσες της ημερομηνίας (σχήμα 3.2.1), αποτελούν μέλη της δομής BCDDate. Η δομή αυτή έχει οριστεί για την κάλυψη των αναγκών του Ημερολογίου στην αποθήκευση και αναζήτηση εγγραφών καθώς και για την παροχή ενός κοινού τρόπου ανταλλαγής ημερομηνιών μεταξύ των δομικών στοιχείων της υλοποίησης. Ο ορισμός της είναι ο ακόλουθος:

```
typedef struct {
    uint8_t year;
    uint8_t mon;
    uint8_t date;
    uint8_t hour;
    uint8_t min;
    uint8_t sec;
} BCDDate;
```

Τα μέλη της έχουν οριστεί σε φυλίνουσα σειρά σπουδαιότητας με τρόπο που διευκολύνεται η σύγχριση μεταξύ ημερομηνιών αυτής της δομής. Αρχικά συγχρίνεται το πρώτο – πλέον σημαντικό – Byte κάθε ημερομηνίας, δηλαδή τα έτη. Εφόσον είναι ίσα, ακολουθεί σύγχριση με το αμέσως επόμενο, σε σπουδαιότητα, Byte, αυτό του μήνα, και η διαδικασία συνεχίζεται για όλα τα Byte ή έως ότου εντοπιστεί κάποια διαφοροποίηση, η οποία κρίνει και το ποια ημερομηνία είναι μεγαλύτερη. Η διαδικασία σύγχρισης που ακολουθείται είναι ίδια με αυτήν μεταξύ δύο μη προσημασμένων αριθμών μεγάλου μήκους.

### 3.2.2 Ομάδα εγγραφών

Τυπικά, δοθέντων δύο ημερομηνιών, το Ημερολόγιο είναι σε θέση να επιστρέψει όλες τις ενδιάμεσες εγγραφές. Ωστόσο, η άμεση και μαζική επιστροφή τους είναι απαγορευτική λόγω των περιορισμένων πόρων του μικροελεγκτή καθώς απαιτείται δέσμευση χώρου στην κύρια μνήμη, δυνητικά, για όλες τις εγγραφές του Ημερολογίου.

Αντιθέτως, η προσπέλαση των εγγραφών πραγματοποιείται σε βήματα. Αρχικά, εκτελείται αναζήτηση βάσει των επιθυμητών ημερομηνιών από όπου επιστρέφεται το πλήθος των εγγραφών που εντοπίστηκαν, καθώς και μία δομή LogRecordSet. Η δομή αυτή αναπαριστά το σύνολο (ή ομάδα) των εντοπισμένων εγγραφών. Ωστόσο, στην πραγματικότητα, περιέχει μόνο την πληροφορία που επιτρέπει να εξαχθεί η επόμενη εγγραφή. Για την οποιαδήποτε επενέργεια στις πραγματικές εγγραφές, παρέχονται ξεχωριστές συναρτήσεις, κάθε μία δεχόμενη τη δομή LogRecordSet. Με τον τρόπο αυτό γίνεται γνωστή η κατάσταση του τρέχοντος συνόλου ώστε να

προσαρμόζεται καταλλήλως η συμπεριφορά της εκάστοτε συνάρτησης, ενώ δίνεται η δυνατότητα ενημέρωσης της δομής μετά την ολοκλήρωση των εργασιών.

### 3.2.3 Οργάνωση εγγραφών

Στο σημείο αυτό αναλύεται πώς διαχειρίζεται το Ημερολόγιο τον αφιερωμένο αποθηκευτικό χώρο για τις εγγραφές του. Παρότι περιγράφονται δύο σχεδιασμοί, ωστόσο, αποτελούν όψεις του ίδιου νομίσματος· και οι δύο υλοποιούνται σε κάθικα και λειτουργούν ο ένας πάνω από τον άλλο, προσδιδόντας διαφορετικά επίπεδα αφαίρεσης. Στο πρώτο κομμάτι περιγράφεται ενώ στο δεύτερο, πώς υλοποιείται η διαχείριση των εγγραφών σε φυσικό επίπεδο.

#### Επίπεδο κυλιόμενου πίνακα

Σε υψηλό αφαιρετικό επίπεδο, ο αποθηκευτικός χώρος νοείται ως ένας μονοδιάστατος πίνακας, εκτεταμένος σε διαδοχικές θέσεις μνήμης, κάθε στοιχείο του οποίου είναι μία εγγραφή. Οι εγγραφές τοποθετούνται σε αύξουσα σειρά βάσει της ημερομηνίας τους, με την παλαιότερη να βρίσκεται πάντα στην πρώτη θέση του πίνακα.

Υπό φυσιολογικές συνθήκες, κάθε νέα εγγραφή διαθέτει ημερομηνία μεγαλύτερη των υπαρχόντων και, συνεπώς, προστίθεται μετά την τρέχουσα τελευταία εγγραφή. Ωστόσο, η τροποποίηση των ρυθμίσεων της συσκευής – για την ακρίβεια της ημερομηνίας/ώρας – είναι πιθανό να προκαλέσει την παραγωγή νέων εγγραφών με ημερομηνία που προηγείται ορισμένων αποθηκευμένων εγγραφών. Η νέα εγγραφή διατηρείται πάντα, σύμφωνα με την πεποίθηση ότι οι νέες ρυθμίσεις που προκαλούν αυτήν τη χρονική ασυνέχεια, έχουν γίνει με σκοπό τη διόρθωση μίας εσφαλμένης κατάστασης της συσκευής. Το ερώτημα τίθεται για τις παλαιότερα αποθηκευμένες εγγραφές που διαθέτουν, πλέον, νεότερη ημερομηνία σε σχέση με κάποια νέα εγγραφή.

Εάν διατηρούνται, τότε μετρήσεις που έχουν γίνει σε παρελθοντικό χρόνο αναμιγνύονται με τις νέες μετρήσεις, ενδεχομένως, νοθεύοντας τα αποτελέσματά τους, εφόσον υπάρχουν αποκλίσεις μεταξύ παλαιών και νέων. Για την αποφυγή τέτοιων αβεβαιοτήτων, το Ημερολόγιο απορρίπτει τις προϋπάρχουσες εγγραφές, η ημερομηνία των οποίων έπειται κάποιας νέας, προς αποθήκευση, εγγραφής.

Οι επιλογές που έχουν περιγραφεί μέχρι στιγμής, καθιστούν δυνατή την προσπέλαση υπαρχόντων και την εισαγωγή νέων εγγραφών σε σταθερό χρόνο, ενώ επιτρέπουν την εύρεση εγγραφών βάσει ημερομηνίας σε χρόνο  $O(\log n)$  (μέσω δυαδικής αναζήτησης). Και τα δύο είναι χαρακτηριστικά που εγγυούνται μειωμένο πλήθος προσβάσεων στη συσκευή μόνιμης αποθήκευσης, η οποία, κατά πάσα πιθανότητα, χαρακτηρίζεται από υψηλότερους χρόνους προσπέλασης. Ένα δεύτερο

πλεονέκτημα είναι η κατά το μέγιστο αξιοποίηση του αφιερωμένου αποθηκευτικού χώρου καθώς αποθηκεύονται μόνο πραγματικά δεδομένα και όχι δευτερεύοντα βιοηθητικά (όπως, για παράδειγμα, δείκτες επόμενου στοιχείου στην περίπτωση χρήσης συνδεδεμένης λίστας). Βέβαια, όλα αυτά είναι άμεση απόρροια της έλλειψης ανάγκης για ενημέρωση και διαγραφή εγγραφών.

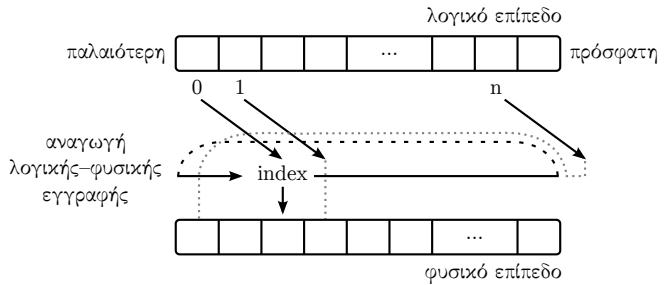
Ένα τελευταίο χαρακτηριστικό του Ημερολογίου είναι η αντιμετώπιση της εξάντλησης του διαθέσιμου αποθηκευτικού χώρου. Σε αυτήν την περίπτωση, επιλέγεται η απόρριψη της παλαιότερης εγγραφής με την (νοητή) μετακίνηση του πίνακα κατά μία θέση ώστε να δημιουργηθεί μία νέα θέση στο τέλος του (εξού και ο χαρακτηρισμός ως «κυλιόμενος»). Σαφώς, ο αντίστοιχος μηχανισμός αναλαμβάνεται από χαμηλότερο επίπεδο της υλοποίησης. Ωστόσο, αξίζει να σημειωθεί ότι στο μεγαλύτερο κομμάτι της υλοποίησης ο αποθηκευτικός χώρος αντιμετωπίζεται με αυτόν τον τρόπο· ως ένας πίνακας όπου το πρώτο του στοιχείο είναι η παλαιότερη εγγραφή, το πραγματικό περιεχόμενο της οποίας αλλάζει ανά πάσα στιγμή, και ότι οι εγγραφές εκτείνονται σε αύξουσα σειρά και καλύπτουν μερικώς ή πλήρως τις θέσεις του πίνακα.

### Επίπεδο κυκλικής ουράς

Επειδή κύριες σημασίες είναι η διατήρηση των τελευταίων μετρήσεων, σε περίπτωση εξάντλησης του διαθέσιμου αποθηκευτικού χώρου, οι νέες εγγραφές αντικαθιστούν τις παλαιότερες. Η συμπεριφορά αυτή μπορεί να αποδοθεί από μία απλουστευμένη μορφή δακτυλίου (ή κυκλικής ουράς), όπου επιτρέπεται μόνο η εισαγωγή στοιχείων στη δομή με υποστήριξη επικάλυψης των παλαιοτέρων. Βέβαια στην πραγματικότητα, ο χώρος αποθήκευσης είναι ένας πίνακας από διαδοχικά Byte ή, λίγο πιο αφαιρετικά, από διαδοχικές θέσεις μεγέθους LogRecord. Προκειμένου να υλοποιηθεί η συμπεριφορά του δακτυλίου, απαιτείται η τήρηση ενός δείκτη που προσδιορίζει τη θέση του πρώτου στοιχείου καθώς και ενός δεύτερου δείκτη για την τελευταία (Κοΐλιας, 2004, σ. 131). Ωστόσο, αντί για δείκτη τέλους, τηρείται το πλήθος των διαθέσιμων εγγραφών. Αυτό έχει το πλεονέκτημα ότι είναι άμεσα γνωστό το πλήθος των εγγραφών χωρίς να απαιτούνται αλγεβρικές πράξεις για την εξαγωγή του και, χυρίως, διευκολύνει την αναγνώριση ενός πλήρως γεμάτου από έναν πλήρως άδειο δακτύλιο (περιπτώσεις κατά τις οποίες οι δείκτες αρχής και τέλους έχουν την ίδια τιμή).

Στο σχήμα 3.2.2 παρουσιάζεται πώς αντιστοιχίζονται αμφιμονοσήμαντα τα στοιχεία του ιδεατού κυλιόμενου πίνακα (λογικό επίπεδο) με τα στοιχεία όπως αυτά είναι πραγματικά αποθηκευμένα στην κυκλική ουρά (φυσικό επίπεδο). Σημειώνεται το εμφανές, ότι για την αναγωγή από το ένα στοιχείο στο άλλο απαιτείται σταυλερός χρόνος.

Σχήμα 3.2.2: Αναγωγή ιδεατής θέσης εγγραφής σε φυσική διεύθυνση.



### 3.3 Όρα συστήματος

Η υλοποίηση περιλαμβάνει τη χρήση ενός ολοκληρωμένου ως ρολόι πραγματικού χρόνου (Real-Time Clock - RTC), το DS1307, το οποίο παρέχει ημερομηνία και ώρα μέχρι δευτερολέπτων, χαρακτηρίζεται από καλή ακρίβεια (απώλεια μερικών δευτερολέπτων το μήνα), χαμηλή κατανάλωση ισχύος και δυνατότητα διατήρησης της λειτουργίας του με εφεδρική τροφοδοσία (μέσω μπαταρίας).

Το RTC διαθέτει 7 καταχωρητές για την ημερομηνία/ώρα και έναν για τη ρύθμιση του παραγόμενου τετραγωνικού παλμού, ενώ ορισμένα bit ελέγχου βρίσκονται σε ορισμένες θέσεις των καταχωρητών ημερομηνίας/ώρας (Maxim, 2008a, σ. 8). Η ημερομηνία/ώρα αναπαριστάται με συμβολισμό BCD (packed Binary-Coded Decimal) σύμφωνα με τον οποίο κάθε δεκαδικό ψηφίο αποθηκεύεται σε 4bit (Maxim, 2008a, σ. 8). Για παράδειγμα, ο αριθμός 12 του δεκαδικού συστήματος αρίθμησης αποκτά δυαδική αναπαράσταση 0001 0002 αντί της συνήθης 0000 1100.

Κατά την αρχική σύνδεση με τη τροφοδοσία, το ρολόι είναι απενεργοποιημένο, κατί που σηματοδοτείται από την ένδειξη CH (Clock Halt) του καταχωρητή 0x00 (Maxim, 2008a, σ. 8). Η επαναφορά του σε 0 ενεργοποιεί το ρολόι και μπορεί να πραγματοποιηθεί παράλληλα με την αρχική ενημέρωση της ημερομηνίας/ώρας.

Η επικοινωνία του RTC με το μικροελεγκτή πραγματοποιείται μέσω διαύλου I<sup>2</sup>C (Inter-Integrated Circuit) σύμφωνα με τον οποίο ο μικροελεγκτής (master του διαύλου) αποστέλλει το bit START ακολουθούμενο από τη διεύθυνση του RTC (στην προχειμένη, 0x68) και το bit πρόθεσης R/W (βλ. Δίαυλος I<sup>2</sup>C (TWI) σ. 46) (Maxim, 2008a, σσ. 1,10).

Κατόπιν, ακολουθεί η διεύθυνση του καταχωρητή προς ανάγνωση/εγγραφή (του RTC) και το επιψημέτο πλήθος Byte. Σαφώς, παρεμβάλλονται τα bit ACK μετά την κάθε επιτυχούσα παραλαβή Byte. Το σχήμα 3.3.1 παρουσιάζει την αποστολή της επιψημητής διεύθυνσης του RTC (Word Address (n)), την επανεκάνηση της επικοινωνίας (Sr) σε λειτουργία ανάγνωσης και την παραλαβή ενός πλήθους

Σχήμα 3.3.1: Παράδειγμα εγγραφής και ανάγνωσης από διεύθυνση του RTC.

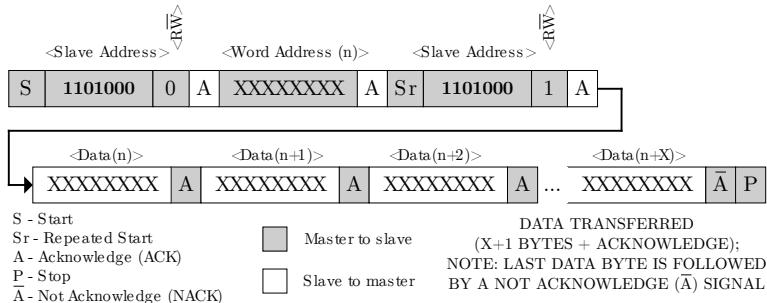


Figure 6. Data Read (Write Pointer, Then Read)—Slave Receive and Transmit. Στο Maxim Integrated. DS1307. 64 x 8, Serial, I<sup>2</sup>C Real-Time Clock. 2008. 14 σσ. url: <http://datasheets.maximintegrated.com/en/ds/DS1307.pdf> (επίσκεψη 11/06/2014), σ. 13

Byte από το RTC. Σημειώνεται ότι το RTC DS1307 υποστηρίζει ανάγνωση/εγγραφή σε burst mode, σύμφωνα με την οποία ο εσωτερικός δείκτης της επιλυμητής διεύθυνσης αυξάνεται αυτόματα με τη διεκπεραίωση κάθε Byte (Maxim, 2008a, σ. 12).

### 3.3.1 Μνήμη RAM

Το RTC DS1307 διαθέτει κοινόχρηστη μνήμη 56Byte η οποία διατηρεί τα δεδομένα της ακόμα και όταν το RTC μεταπίπτει στην εφεδρική τροφοδοσία (μπαταρία) (Maxim, 2008a, σ. 1). Το χαρακτηριστικό αυτό είναι ιδιαίτερα χρήσιμο, καθώς η μνήμη του μπορεί να χρησιμοποιηθεί για την αποθήκευση ρυθμίσεων της συσκευής που διατηρούνται μεταξύ διαδοχικών αποσυνδέσεών της από την κύρια τροφοδοσία που, ωστόσο, είναι εύκολο να αναιρεθούν με την προσωρινή αποσύνδεση της μπαταρίας από τη συσκευή (προκαλώντας επαναφορά στις εργοστασιακές της ρυθμίσεις) (βλ. Εφεδρική μνήμη σ. 24).

Οι διευθύνσεις της μνήμης έπονται των καταχωρητών (διεύθυνση 0x08) και εκτείνονται μέχρι την 0x3F (Maxim, 2008a, σ. 8).

## 3.4 Εργασία

Στο κεφάλαιο του υποσυστήματος κίνησης (σ. 85) παρουσιάζεται ο υποκείμενος μηχανισμός για τον έλεγχο της κίνησης της κεφαλής της συσκευής. Ωστόσο, στο πλαίσιο της υλοποίησης, η χρήση του υποσυστήματος κίνησης είναι άρρηκτα συνδεδεμένη με την πραγματοποίηση μετρήσεων. Η κεφαλή μετακινείται σε μία θέση με σκοπό τη δειγματοληψία και καταχώρηση της μέτρησης των αισθητήρων για εκείνο το σημείο του υλικού. Αυτή η πρόσθετη λειτουργία παρέχεται από

ξεχωριστό τμήμα, τη μονάδα εργασίας (task).

### 3.4.1 Δειγματοληψία

Η πραγματοποίηση μίας εργασίας για τη δειγματοληψία του μέσου μπορεί να περιγραφεί από τα ακόλουθα στάδια:

1. Μετατόπιση κεφαλής σε νέο ζεύγος συντεταγμένων XY.
2. Διείσδυση της κεφαλής στο παρακολουθούμενο μέσο.
3. Λήψη μέτρησης των αισθητήρων και καταχώρησή τους στο Ημερολόγιο (σ. 25).
4. Ανύψωση της κεφαλής.

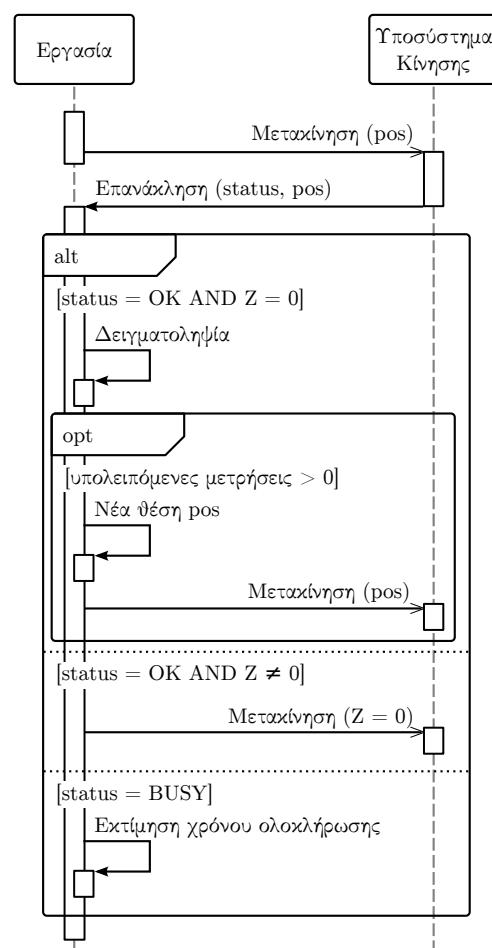
Δεδομένου ότι η κεφαλή βρίσκεται ανυψωμένη (προεπιλεγμένη θέση κατόπιν παλινόρθησης) (βλ. σ. 101), τα δύο πρώτα στάδια πραγματοποιούνται αυτόματα από τον υποκείμενο μηχανισμό. Κατά την ολοκλήρωση της μετατόπισης της κεφαλής (συμπεριλαμβανομένης και της διείσδυσής στο μέσο), το υποσύστημα κίνησης ενημερώνει για το συμβάν (μέσω επανάλησης) το υποσύστημα εργασίας (ή όποιο άλλο του έχει δηλωθεί).

Στο σημείο αυτό, η μονάδα εργασίας αναλαμβάνει το 3<sup>o</sup> στάδιο. Κατόπιν, προκαλεί την ανύψωση της κεφαλής, η οποία εκτελείται ασύγχρονα (όπως συμβαίνει, άλλωστε, με κάθιση της κεφαλής). Η μονάδα εργασίας ειδοποιείται ξανά όταν η κεφαλή έχει επανατοποιηθεί στην κορυφαία της θέση. Πλέον, εφόσον απαντείται, είναι δυνατό να δρομολογηθεί μία νέα μέτρηση προκαλώντας τη μετατόπιση της κεφαλής σε νέες συντεταγμένες XYZ με Z = 0, ώστε το υποσύστημα εργασίας να ειδοποιηθεί όταν η κεφαλή βρίσκεται σε θέση για νέα δειγματοληψία.

Στο σχήμα 3.4.1 παρουσιάζεται πώς το υποσύστημα εργασίας προκαλεί την αρχική ενεργοποίηση τους υποσυστήματος κίνησης (για τη μετακίνηση σε νέα θέση) και πώς οι διαδοχικές διεκπεραιώσεις της μετακίνησης της κεφαλής οδηγούν, σταδιακά, στην ολοκλήρωση του ανατεθειμένου πλήθους μετρήσεων. Η ενεργοποίηση της εργασίας παραλείπεται το σχήμα, εσκεμμένα, και ο λόγος περιγράφεται στην Εκκίνηση εργασίας (σ. 35).

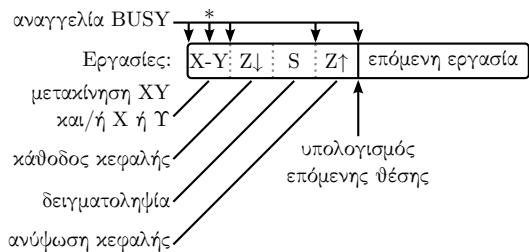
Ένα αναγκαίο στοιχείο για την πραγματοποίηση πολλαπλών μετρήσεων είναι η ύπαρξη ενός μηχανισμού για την παραγωγή νέων, προς δειγματοληψία, θέσεων. Ένας τέτοιος μηχανισμός θα μπορούσε να λαμβάνει υπόψη προηγούμενες μετρήσεις ώστε να προτιμώνται τοποθεσίες που έχουν καιρό να υποστούν εξέταση ή εκείνες στις οποίες είχαν εντοπιστεί κρίσιμες τιμές. Ωστόσο, στο πλαίσιο της υλοποίησης, ο μηχανισμός αρκείται στην παραγωγή τυχαίων θέσεων που βασίζονται, ελαφρώς, στην τρέχουσα ημέρα και ώρα της συσκευής και στην τρέχουσα θέση της κεφαλής.

Σχήμα 3.4.1: Σταδιακή ολοκλήρωση φόρτου εργασίας μετρήσεων.



### Σχήμα 3.4.2: Στάδια περάτωσης μίας εργασίας μέτρησης.

Οι στιγμές αναγγελίας BUSY (βέλη) αποτελούν τα σημεία όπου ανανεώνεται η εκτίμηση του χρόνου ολοκλήρωσης. Ο αστερίσκος (\*) δηλώνει μία ενδεχόμενη πρόσθιτη αναγγελία, εφόσον υπολείπεται κινηση σε έναν εκ των δύο αξόνων X ή Y.



### Εκτιμώμενος χρόνος ολοκλήρωσης

Στο σχήμα 3.4.1 γίνεται αναφορά σε εκτίμηση του χρόνου ολοκλήρωσης η οποία εκτελείται όποτε η επανάληση αναφέρει ότι το υποσύστημα κίνησης είναι απασχολημένο (BUSY). Ο λόγος ύπαρξης του μηχανισμού εκτίμησης είναι η παροχή μίας ένδειξης του χρόνου για τον οποίο το υποσύστημα κίνησης, καθώς και όλες οι λειτουργίες που βασίζονται σε αυτόν, είναι αδύνατο να χρησιμοποιηθούν έως ότου παρέλθει. Χαρακτηριστική περίπτωση χρήσης του εκτιμώμενου χρόνου είναι στο πεδίο κεφαλίδας HTTP Retry-After κατά τις αποχρίσεις με κωδικούς κατάστασης 202 και 503 (βλ. Πόροι υλοποίησης σ. 133).

Όπως αναφέρεται στο Υποσύστημα Κίνησης (σσ. ??,95), η μετατόπιση της κεφαλής σε νέα θέση πραγματοποιείται σε τρία, το πολύ, στάδια. κοινή μετατόπιση στο επίπεδο X-Y, υπολειπόμενη μετατόπιση σε άξονα X ή Y και μετατόπιση σε άξονα Z ή, εναλλακτικά, πρώτα η μετατόπιση στον άξονα Z και μετά των άλλων δύο. Πριν την εκκίνηση κάθιση σταδίου, και ενώ οι κινητήρες βρίσκονται σε ηρεμία, αναγγέλλεται από το υποσύστημα κίνησης (μέσω της επανάλησης) ο νέος προορισμός και η αλλαγή της κατάστασής του σε BUSY. Η πληροφορία αυτή σε συνδυασμό το υπολειπόμενο πλήθος μετρήσεων μπορεί να χρησιμοποιηθεί για την εκτίμηση της συνολικής διάρκειας της εργασίας.

Το σχήμα 3.4.2 παρουσιάζει τις συνιστώσες της εργασίας καθώς και τις στιγμές που αναγγέλλεται η ολοκλήρωση ενός μέρους της μετατόπισης (σημεία BUSY). Στο σχήμα, στην περίπτωση της μετατόπισης στο επίπεδο X-Y, σημειώνεται μία ενδεχόμενη δεύτερη αναγγελία (με αστερίσκο) όταν πραγματοποιείται ξεχωριστή μετατόπιση σε έναν εκ των δύο αξόνων για το υπολειπόμενο πλήθος βημάτων. Η λεπτομέρεια αυτή είναι αδιάφορη για τη μονάδα εργασίας, καθώς αυτό που ενδιαφέρει σε σχέση με το επίπεδο X-Y, είναι η μέγιστη μετατόπιση μεταξύ των δύο αξόνων.

Όπως αναφέρεται προηγουμένως, η νέα θέση της κεφαλής στο επίπεδο X-Y υπολογίζεται τη στιγμή ολοκλήρωσης της προηγούμενης εργασίας. Αυτή η σχεδιαστική επιλογή καθιστά αδύνατο τον ακριβή υπολογισμό του χρόνου ολοκλήρωσης του συνόλου των εργασιών καθώς οι θέσεις που, τελικά, θα προκύψουν είναι άγνωστες σε προγενέστερη στιγμή της εκτέλεσης.

Για το λόγο αυτό χρησιμοποιείται ένας αναμενόμενος χρόνος μετατόπισης της κεφαλής στο επίπεδο X-Y για τις υπολειπόμενες εργασίες ο οποίος, όσο μεγαλύτερο το πλήθος εργασιών, τόσο μεγαλύτερη απόκλιση από τον πραγματικό χρόνο που τελικά θα απαιτηθεί. Ωστόσο, καθώς ολοκληρώνονται τα επιμέρους στάδια κάθε εργασίας και πραγματοποιείται νέα εκτίμηση (σημεία αναγγελίας BUSY), ο εκτιμώμενος χρόνος γίνεται ολοένα πιο έγκυρος. Η απόκλιση της εκτίμησης του χρόνου ολοκλήρωσης της τελευταίας εργασίας κυμαίνεται στα 2, περίπου, δευτερόλεπτα, μετρική αποδεκτή για τις απαιτήσεις της υλοποίησης.

Η εκτίμηση του χρόνου ολοκλήρωσης του φόρτου εργασίας καθώς και η ώρα κατά την οποία πραγματοποιήθηκε η εκτίμηση, διατηρούνται. Όποτε προκύπτει ανάγκη για την επιστροφή του αναμενόμενου χρόνου ολοκλήρωσης ξεκινώντας κάποια δεδομένη χρονική στιγμή, αρκεί να αφαιρεθεί από εκείνη την ώρα, η ώρα της τελευταίας εκτίμησης και να συγχριθεί το αποτέλεσμα με την εκτίμηση χρόνου για να διαπιστωθεί πόσος εκτιμημένος χρόνος έχει παρέλθει.

### Εκκίνηση εργασίας

Η δειγματοληψία του μέσου είναι δυνατό να ξεκινήσει με δύο τρόπους: κατόπιν προτροπής εξωτερικής οντότητας ή από το ίδιο το σύστημα υπό τις κατάλληλες συνθήκες. Η εκκίνηση εργασιών από εξωτερική οντότητα περιγράφεται στη μέθοδο POST του πόρου μετρήσεων του διαχομιστή (σ. 139) και παρέχεται κυρίως για λόγους εξαχρίβωσης λειτουργίας.

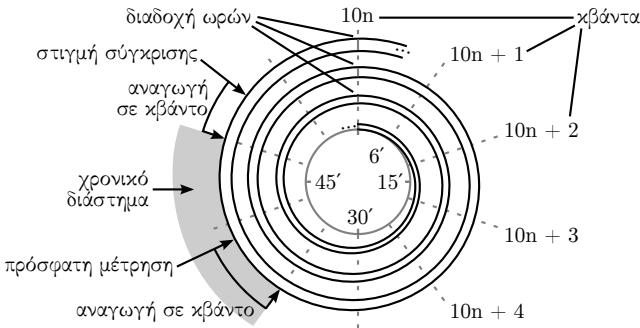
Η κύρια πηγή εκκίνησης των κύκλων εργασιών παραμένει το ίδιο το σύστημα. Για το σκοπό αυτό, χρίνεται αρκετή η τήρηση της ώρας της πιο πρόσφατης μετρησης και μίας ένδειξης για το ελάχιστο επιθυμητό διάστημα μεταξύ διαδοχικών κύκλων εργασίας. Επίσης, απαιτείται η ύπαρξη ενός ελέγχου για την εξαχρίβωση του χρόνου που έχει παρέλθει από την πλέον πρόσφατη μέτρηση μέχρι τη στιγμή που πραγματοποιείται ο έλεγχος.

Προφανώς, η ημερομηνία της πλέον πρόσφατης μέτρησης τίθεται κατά την αρχικοπόίηση της συσκευής (power-on) βάσει της τελευταίας καταχωρημένης εγγραφής του ημερολογίου και ενημερώνεται με κάθε νέα πραγματοποιηθείσα μέτρηση.

Ωστόσο, στην πραγματικότητα, ο μηχανισμός βασίζεται στη χρήση χρονοσφραγίδων που αντιπροσωπεύουν τα λεπτά που έχουν παρέλθει από την αρχή της ημέρας για κάθε ημερομηνία. Για την αχρίβεια, επιλέγεται η ημέρα να χωρίζεται

**Σχήμα 3.4.3:** Αντιστοίχηση ωρών σε κβάντα και υπολογισμός χρονικού διαστήματος.

Με τη συμβολίζεται η κάθε ώρα της ημέρας και χυμαίνεται μεταξύ 0 και 23, ενώ για κάθε ώρα ορίζονται 10 κβάντα. Συνολικά, μία μέρα διαθέτει 240 κβάντα.



σε 240 κβάντα, ενώ κάθε χρονοσφραγίδα προσδιορίζει κάποια στιγμή της ημέρας ως ένα πλήθος τέτοιων κβάντων. Όποτε προκύπτει ανάγκη για τον έλεγχο του χρονικού διαστήματος που έχει παρέλθει μεταξύ πρόσφατης και τρέχουσας ώρας, η τρέχουσα ώρα ανάγεται σε χρονοσφραγίδα (δηλαδή, σε κβάντα των 6 λεπτών) και συγχρίνεται με τη χρονοσφραγίδα της μέτρησης (σχήμα 3.4.3).

Ο υπολογισμός της χρονοσφραγίδας είναι απλός: οι ώρες μετατρέπονται σε λεπτά και χωρίζονται σε κβάντα, κατόπιν, προστίθεται η αξία (ευκλείδεια διαίρεση) των λεπτών σε κβάντα, ως εξής:

$$Q = \frac{60hrs + min}{q} \quad (3.4.1)$$

, όπου  $q$  η αξία κάθε κβάντου σε λεπτά ( $q = 6$ ).

Ως άμεσο αποτέλεσμα, είναι δυνατή η διάχριση των ωρών με ακρίβεια το πολύ 6 λεπτών. Επιπλέον, η τήρηση μέχρι 240 κβάντων (δηλαδή, μίας ημέρας) συνεπάγεται ότι οι δύο ημερομηνίες συγχρίνονται μόνο ως προς τις ώρες και τα λεπτά, αγνοώντας τη διαφορά τους σε ημέρες, μήνες και έτη. Τυπικά, το πρόβλημα αυτό είναι αμελητέο καθώς η σύγχριση των χρονοσφραγίδων πραγματοποιείται κάθε λίγα δευτερόλεπτα. Μόνο στην περίπτωση παρατεταμένης διακοπής της τροφοδοσίας του ρεύματος ενδέχεται να εμφανιστεί κάποια ασυνέχεια με τη μορφή καινούστερημένης εκκίνησης του επόμενου κύκλου εργασίας που οφείλεται στο ότι η συσκευή αναμένει να παρέλθει το καθορισμένο πλήθος κβάντων από την τελευταία ώρα μέτρησης, ανεξαρτήτως της ημέρας κατά την οποία πραγματοποιήθηκε.

### Περιοδικός έλεγχος

Είναι σαφές ότι ο έλεγχος της χρονοσφραγίδας απαιτεί ένα έναυσμα που να προκαλεί την εκτέλεσή του. Επίσης, όταν πρέπει να είναι ικανό να αφυπνίζει τη μονάδα επεξεργασίας από λειτουργία χαμηλής κατανάλωσης που αυτή τίθεται όσο αναμένει κάποια νέα εργασία. Σύμφωνα με το εγχειρίδιο της Atmel (2013, σ. 38), ο χρονομετρητής WDT (Watchdog Timer) μπορεί να χρησιμοποιηθεί για την αφύπνιση της MCU από οποιαδήποτε λειτουργία χαμηλής κατανάλωσης.

Ο χρονομετρητής WDT χρησιμοποιεί πολλούς ανεξάρτητους ταλαντωτή και ρυθμίζεται για περιοδική ενεργοποίηση (μέσα από ένα εύρος πιθανών περιόδων) κατά την οποία προκαλεί διακοπή ή ακόμα και επανεκκίνηση του μικροελεγκτή (reset) (Atmel, 2013, σ. 50). Το δεύτερο είναι ιδιαίτερα σημαντικό για την αποφυγή ατερμόνων βρόχων αλλά απαιτεί την, από λογισμικού, επανέναρξη του μετρητή πριν την πρόκληση της επανεκκίνησης (Atmel, 2013, σ. 50).

Στο πλαίσιο της υλοποίησης, ο WDT ρυθμίζεται μόνο για την αναγγελία διακοπής, η οποία είτε αφυπνίζει την MCU και εκτελεί τη ρουτίνα εξυπηρέτησης της, είτε αναμένει την ολοκλήρωση κάποιας άλλης ρουτίνας (για παράδειγμα, την απόχριση σε κάποιο εισερχόμενο αίτημα HTTP) πριν εκτελεστεί η ίδια για την εκκίνηση του επόμενου κύκλου εργασίας, εφόσον απαιτείται. Σημειώνεται ότι ο επιλεγμένος μικροελεγκτής (ATmega328P), υποστηρίζει περιόδους από 16ms μέχρι 8s (Atmel, 2013, σ. 55).

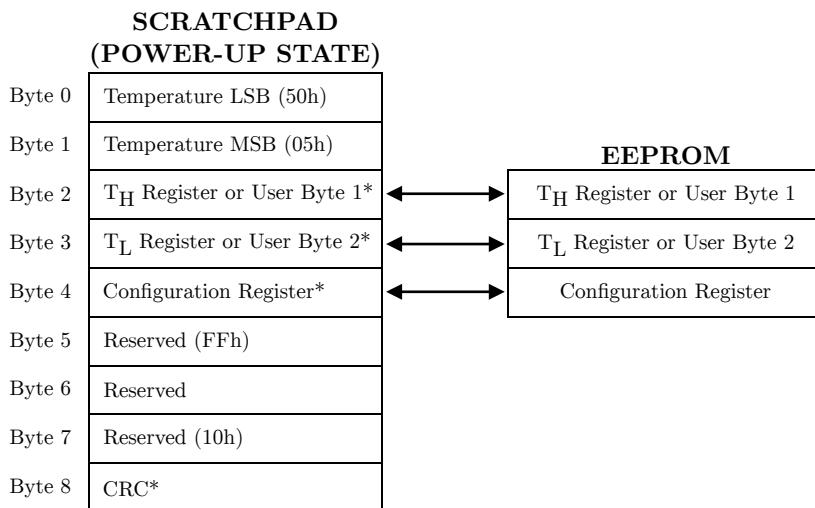
#### 3.4.2 Μέτρηση θερμοκρασίας

Στο παρόν στάδιο της υλοποίησης, μόνο η μελέτη και διασύνδεση με τον αισθητήρα θερμοκρασίας υλοποιείται. Η μέτρηση της θερμοκρασίας αναλαμβάνεται από ψηφιακό θερμόμετρο, το ολοκληρωμένο DS18B20, το οποίο βρίσκεται ενσωματωμένο σε στεγανό σωληνάριο και προσαρτημένο στην κινητή κεφαλή της συσκευής ώστε καθώς αυτή κινείται να επιτρέπεται η μεταφορά του γύρω και εντός του παρακολουθούμενου υλικού. Το επιλεγμένο ολοκληρωμένο χαρακτηρίζεται από ακρίβεια  $\pm 0.5^{\circ}\text{C}$  σε θερμοκρασία λειτουργίας  $-55$  έως  $+125^{\circ}\text{C}$ , διασύνδεση 1-Wire (βλ. σ. 42) με δυνατότητα τροφοδοσίας από τη (μοναδική) γραμμή του διαύλου με ρυθμιζόμενη ακρίβεια από 9 έως 12bit (Maxim, 2008b, σ. 1).

#### Καταχωρητές DS18B20

Η μνήμη του αποτελείται από 9Byte με πλέον σημαντικά, για τις ανάγκες της υλοποίησης, τα δύο πρώτα, στα οποία εναποτίθεται η μέτρηση της θερμοκρασίας. Ακολουθούν δύο καταχωρητές που προσδιορίζουν δύο όρια θερμοκρασιών τα οποία αν ξεπεραστούν από κάποια μέτρηση, θέτουν σχετική ένδειξη στην εσωτερική κα-

Σχήμα 3.4.4: Πρητική μνήμη (scratchpad) και μνήμη EEPROM του DS18B20.



\* Power-up state depends on value(s) stored in EEPROM.

Βασισμένο Figure 7. DS18B20 Memory Map. Στο Maxim Integrated. DS18B20 Programmable Resolution 1-Wire® Digital Thermometer. 22 Απρ. 2008. 27 σσ. url: <http://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html> (επίσκεψη 10/03/2014), σ. 7

τάσταση του ολοκληρωμένου, η οποία μπορεί να χρησιμοποιηθεί για τη συμμετοχή στη διαδικασία επιλογής slave συσκευής, μόνο εκείνων των ολοκληρωμένων που χρήζουν άμεσης ανάγκης (Maxim, 2008b, σσ. 4–5). Προφανώς, συγκρίνεται μόνο το ακέραιο τμήμα της θερμοκρασίας με την τιμή των καταχωρητών καθώς οι καταχωρητές είναι του 1Byte ο καθένας (βλ. και σχήμα 3.4.5). Η σχετική εντολή ROM ονομάζεται ALARM SEARCH. Εναλλακτικά, μπορούν να χρησιμοποιηθούν για την αποθήκευση δεδομένων της συσκευής που, σε συνδυασμό με την εντολή COPY SCRATCHPAD (βλ. παρακάτω), διατηρούνται μεταξύ διαδοχικών αποσυνδέσεων της συσκευής από την τροφοδοσία.

Ο καταχωρητής στη διεύθυνση 4 καθορίζει την ακρίβεια και, συνεπώς το χρόνο που απαιτεί κάθε μέτρηση (Maxim, 2008b, σσ. 4,8). Το ποστηρίζεται ακρίβεια των 12 έως 9bit, με κάθε μικρότερη να απορρίπτει ένα μέρος του κλάσματος της μέτρησης.

Τα Byte 5 έως 7 χρησιμοποιούνται για τις ανάγκες του ολοκληρωμένου, ενώ στο Byte 8 υπολογίζεται το αποτέλεσμα του πολυωνύμου για την επιβεβαίωση της εγκυρότητας των δεδομένων κατά την ανάγνωσή τους από το master.

Η μορφή με την οποία αποθηκεύεται η θερμοκρασία στους δύο πρώτους καταχωρητές παρουσιάζεται στο σχήμα 3.4.5. Η θερμοκρασία αποθηκεύεται με πρόσημο ως συμπλήρωμα του 2 (Maxim, 2008b, σ. 3). Στο πλαίσιο της υλοποίησης, αυτοί είναι οι μοναδικοί καταχωρητές του DS18B20 που αξιοποιούνται.

Σχήμα 3.4.5: Μορφή θερμοκρασίας στους καταχωρητές 0 και 1 του DS18B20.

bit:	7	6	5	4	3	2	1	0
LS Byte	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$
bit:	15	14	13	12	11	10	9	8
MS Byte	S	S	S	S	S	$2^6$	$2^5$	$2^4$

S = SIGN

Βασισμένο Figure 2. Temperature Register Format. Στο Maxim Integrated. DS18B20 Programmable Resolution 1-Wire® Digital Thermometer. 22 Απρ. 2008. 27 σσ. url: <http://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html> (επίσκεψη 10/03/2014), σ. 4

### Τυποστηριζόμενες εντολές

Στην ενότητα Δίαυλος 1-Wire (σ. 42) περιγράφεται η έννοια της συναλλαγής κατά την οποία πραγματοποιείται αρχικοποίηση, ακολουθεί η επιλογή μία συσκευής slave μέσω εντολών ROM και, τελικά, αποστέλλεται η εντολή λειτουργίας (Function command) που είναι επιθυμητό να εκτελεστεί. Στην περίπτωση του DS18B20, οι υποστηριζόμενες εντολές λειτουργίας είναι οι ακόλουθες:

**CONVERT T [0x44]** Προκαλεί την λήψη μίας νέας μέτρησης. Σε περίπτωση που η τροφοδοσία του ολοκληρωμένου είναι ανεξάρτητη από τη γραμμή του διαιύλου, η ολοκλήρωση σηματοδοτείται από την απελευθέρωση της γραμμής από το DS18B20 και την επαναφορά του σε λογικό 1 από τον αντιστάτη pull-up (Maxim, 2008b, σ. 11). Σε αντίθετη περίπτωση, ο μικροελεγκτής είναι υποχρεωμένος να αποφασίσει πόσο χρόνο να αναμείνει ώστε να ολοκληρωθεί η μετατροπή, με μέγιστο 750ms για ακρίβεια 12bit (Maxim, 2008b, σσ. 8,11).

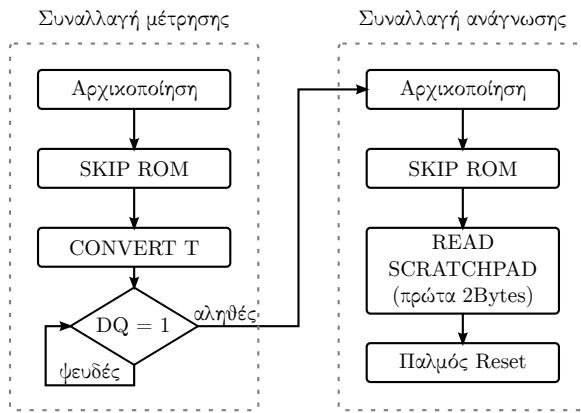
**WRITE SCRATCHPAD [0x4E]** Επιτρέπει την αποστολή τριών Byte που εναποτίθενται στις διευθύνσεις 2, 3 και 4 της πτητικής μνήμης του ολοκληρωμένου (καταχωρητές  $T_H$ ,  $T_L$  και ρυθμίσεων) (Maxim, 2008b, σ. 11).

**READ SCRATCHPAD [0xBE]** Επιτρέπει την πλήρη ή μερική ανάγνωση της πτητικής μνήμης του ολοκληρωμένου ξεκινώντας από το λιγότερο σημαντικό bit του Byte 0, διαδοχικά εκτεινόμενη προς στις επόμενες θέσεις (Maxim, 2008b, σ. 11).

**COPY SCRATCHPAD [0x48]** Μεταφέρει τα Byte  $T_H$ ,  $T_L$  και ρυθμίσεων (Byte 2, 3 και 4, αντιστοίχως) από την πτητική μνήμη του DS18B20 στη μνήμη EEPROM ώστε να διατηρούνται μεταξύ διαδοχικών αποσυνδέσεων του ολοκληρωμένου (Maxim, 2008b, σ. 12).

**Σχήμα 3.4.6:** Συναλλαγές 1-Wire για τη δειγματοληψία της θερμοκρασίας.

Στην υλοποίηση, χρησιμοποιείται μόνο το DS18B20 ως slave συσκευή με αποτέλεσμα να μην απαιτείται η ρητή αναγγελία του αναγνωριστικού του, και για αυτό χρησιμοποιείται η εντολή SKIP ROM. Με DQ συμβολίζεται η κατάσταση της γραμμής που διαιύλου 1-Wire, ενώ ο τερματικός Πλαμός Reset θα μπορούσε, κάλλιστα, να ανήκει σε επόμενο ζεύγος παλμών αφχικοποίησης (βλ. σ. 44).



**RECALL E<sup>2</sup> [0xB8]** Επαναφέρει τα Byte 2, 3 και 4 από τη μνήμη EEPROM στην πτητική μνήμη του DS18B20 (Maxim, 2008b, σ. 12).

**READ POWER SUPPLY [0xB4]** Επιτρέπει στο master να αναγνωρίσει εάν το ολοκληρωμένο τροφοδοτείται μέσω του ακροδέκτη V<sub>CC</sub> ή εάν βασίζεται στη γραμμή του διαιύλου (Maxim, 2008b, σ. 12).

Στο πλαίσιο της υλοποίησης, οι εντολές που χρησιμοποιούνται είναι οι CONVERT T και READ SCRATCHPAD. Μία τυπική επικοινωνία μεταξύ του μικροελεγκτή και του DS18B20 απεικονίζεται στο σχήμα 3.4.6 κατά την οποία πραγματοποιούνται δύο συναλλαγές, μία για την εκκίνηση της μέτρησης και μία για την ανάγνωση της μνήμης και, για την ακρίβεια, των δύο πρώτων καταχωρητών που αντιστοιχούν στη μέτρηση. Αυτό είναι δυνατό να επιτευχθεί αποστέλλοντας τον παλμό Reset αμέσως μετά την ανάγνωση του δεύτερου Byte.

Όπως παρουσιάζεται στο σχήμα, η επιλογή του DS18B20 γίνεται μέσω της εντολής SKIP ROM η οποία, στην πραγματικότητα, παραλείπει την αναγγελία αναγνωριστικού και μεταβαίνει, απευθείας, στην αποστολή εντολής λειτουργίας (Function command). Η συγκεκριμένη τακτική μπορεί να χρησιμοποιηθεί υπό την προϋπόθεση ότι το DS18B20 είναι το μοναδικό ολοκληρωμένο που συνδέεται στο δίαιυλο 1-Wire, κάτι που ισχύει για την υλοποίηση. Σε αντίθετη περίπτωση, η επιλογή περισσότερων ολοκληρωμένων θα προκαλούσε σύγκρουση κατά την ανάγνωση της μέτρησης όπου ταυτόχρονα θα επιχειρούσαν να αναγγείλουν τα bit τους.

Τέλος, αναφέρεται ότι κατόπιν υποβολής της εντολής μέτρησης της θερμοκρασίας, ο μικροελεγκτής αναφέρει την απελευθέρωση (και επαναφορά σε λογικό 1) της γραμμής του διαιύλου από το DS18B20 προκειμένου να προβεί στη συναλλαγή της ανάγνωσης. Η δυνατότητα αναγγελίας της ολοκληρωσης με αυτόν τον τρόπο είναι διαιθέσιμη, όπως αναφέρεται παραπάνω, μόνο στην περίπτωση που το DS18B20 τροφοδοτείται μέσω του ακροδέκτη  $V_{CC}$  και όχι μέσω της ίδιας της γραμμής του διαιύλου.

Στο πλαίσιο της υλοποίησης επιλέγεται η χρήση του ακροδέκτη  $V_{CC}$  για την τροφοδοσία του DS18B20 για δύο λόγους. Καταρχήν, για τη τροφοδοσία του ολοκληρωμένου μέσω της γραμμής, η Maxim (2008b, σ. 5) συνιστά τη χρήση ενός ισχυρού στοιχείου για την επαναφορά της (όπως MOSFET) και όχι μόνο ενός αντιστάτη καθώς κατά την μέτρηση της θερμοκρασίας, η απότομη αύξηση κατανάλωσης έντασης ρεύματος μπορεί να προκαλέσει πτώση στην τάση της γραμμής με αποτέλεσμα η εργασία να μην ολοκληρωθεί επιτυχώς. Με τη σειρά του, η μέθοδος αυτή απαιτεί ένα επιπλέον τρανζίστορ καθώς και την απόδοση ενός επιπρόσθετου ακροδέκτη του μικροελεγκτή για το σκοπό.

Ο δεύτερος λόγος σχετίζεται με τη μορφή του αισθητήριου οργάνου. Το προϊόν που χρησιμοποιείται διαιθέτει ένα DS18B20 ενσωματωμένο στο ένα άκρο ενός στεγανού σωληναρίου με τρεις απολήξεις σύνδεσης στο άλλο.  $V_{CC}$ , GND και DQ. Επομένως, ήδη υπάρχει αγωγός για τη σύνδεση του ακροδέκτη  $V_{CC}$  με την τροφοδοσία με αποτέλεσμα η χρήση της γραμμής DQ για την τροφοδοσία να φαντάζει άσκοπη.

### 3.5 Δίαυλοι επικοινωνίας ολοκληρωμένων

Στο πλαίσιο της υλοποίησης, γίνεται χρήση των ακόλουθων διαιύλων για την επικοινωνία του μικροελεγκτή με επιλεγμένα ολοκληρωμένα κυκλώματα:

**1-Wire** της Dallas Semiconductor, δίαυλος ασύγχρονης ημιαμφίδρομης επικοινωνίας που χρησιμοποιεί μόνο έναν αγωγό (σήμα) διαισθνδεσης. Χρησιμοποιείται για την επικοινωνία με τον αισθητήρα θερμοκρασίας DS18B20. Εφόσον ο μικροελεγκτής αδυνατεί να το υποστηρίξει εγγενώς, η υλοποίηση του διαιύλου γίνεται εξολοκλήρου σε λογισμικό (bit-banging).

**I<sup>2</sup>C** της Phillips, δίαυλος σύγχρονης ημιαμφίδρομης επικοινωνίας που χρησιμοποιεί δύο αγωγούς σύνδεσης: ένα για συγχρονισμό και έναν για δεδομένα. Χρησιμοποιείται για την επικοινωνία με το ρολόι πραγματικού χρόνου (RTC) DS1307. Το λογισμικό οδήγησης αξιοποιεί το υποκείμενο, και συμβατό, κύκλωμα TWI (Two-Wire Interface) του μικροελεγκτή.

**SPI** ονομασμένο από τη Motorola, δίαιυλος σύγχρονης αμφίδρομης επικοινωνίας που χρησιμοποιεί τρεις αγωγούς επικοινωνίας – ρολόι, δεδομένα Master και δεδομένα Slave – καθώς και επιπρόσθετη γραμμή επιλογής ( $\overline{CS}$ ) του εκάστοτε ολοκληρωμένου. Χρησιμοποιείται για την επικοινωνία με το ολοκληρωμένο δικτύωσης W5100 και της εξωτερικής μνήμης Flash. Το λογισμικό οδήγησης κάθισε ολοκληρωμένου αξιοποιεί το υποκείμενο κύκλωμα SPI του μικροελεγκτή.

### 3.5.1 Δίαιυλος 1-Wire

Ο δίαιυλος 1-Wire δημιουργήθηκε από την Dallas Semiconductor και για τη διασύνδεση των συσκευών χρησιμοποιείται μία μόνο γραμμή, η οποία μπορεί να χρησιμοποιηθεί, υπό περιπτώσεις, και για την τροφοδοσία των συσκευών επιπροσθέτως της ανταλλαγής δεδομένων (Atmel, 2004, σ. 1). Η επικοινωνία είναι ημιαμφίδρομη, ενώ για την αποστολή κάθισε bit, ανεξαρτήτως κατεύθυνσης, χρησιμοποιούνται χρονοθυρίδες οι οποίες δημιουργούνται από το μοναδικό master του διαιύλου (Atmel, 2004, σ. 2. Maxim, 2008b, σ. 15).

#### Εγγραφή

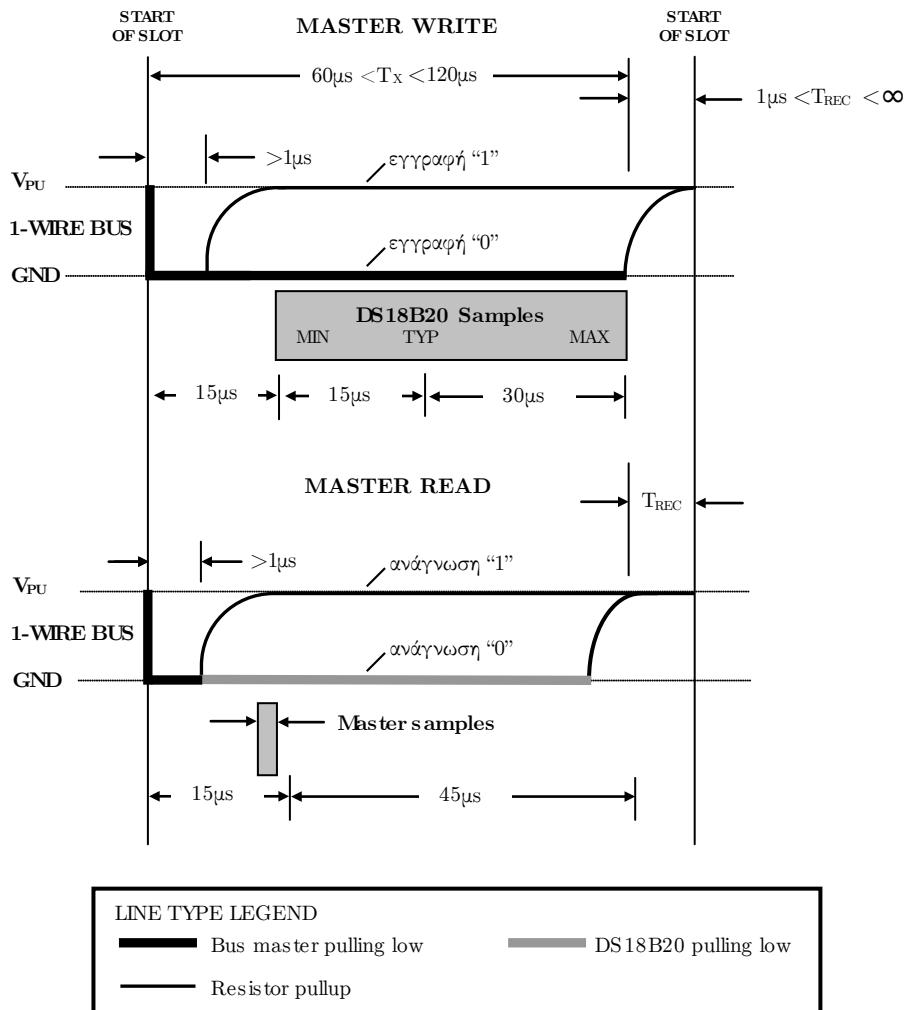
Η μορφή των χρονοθυρίδων εμφανίζεται στο σχήμα 3.5.1. Σε περίπτωση εγγραφής ενός bit, ο master θέτει τη γραμμή σε λογικό 0 για 1 έως 15μs και, κατόπιν, είτε τη διατηρεί σε αυτήν την κατάσταση αναγγέλλοντας, με αυτόν τον τρόπο, bit 0, είτε τίθεται σε κατάσταση υψηλής εμπέδωσης, ώστε ο αντιστάτης pull-up να επαναφέρει τη γραμμή σε λογικό 1 (Atmel, 2004, σ. 2).

#### Ανάγνωση

Για την ανάγνωση ενός bit, ο master θέτει τη γραμμή σε λογικό 0 τουλάχιστον για 1μs και, κατόπιν, τίθεται σε κατάσταση υψηλής εμπέδωσης, δίνοντας τη δυνατότητα στο slave να αναγγείλει το bit, ο οποίος, με τη σειρά του, είτε θέτει τη γραμμή σε λογικό 0, είτε παραμένει σε κατάσταση υψηλής εμπέδωσης (Atmel, 2004, σ. 2). Ο master ελέγχει την κατάσταση της γραμμής εντός 15μs από την αρχή της χρονοθυρίδας (Maxim, 2008b, σ. 17) προκειμένου να ανάγει το bit του slave.

Είναι εμφανές ότι η ανάγνωση είναι παρόμοια με την εγγραφή bit τιμής 1· στην πραγματικότητα, αυτό που διαφοροποιείται είναι η εσωτερική κατάσταση του slave που καθορίζει εάν πρόκειται να λάβει ή να εγγράψει κάποιο bit, όπως, για παράδειγμα, εάν έχει προηγουμένως αποσταλεί εντολή ανάγνωσης κάποιας διεύθυνσης της μνήμης του. Όπως συνιστάται στο εγχειρίδιο της Maxim (2008b, σ. 17) και

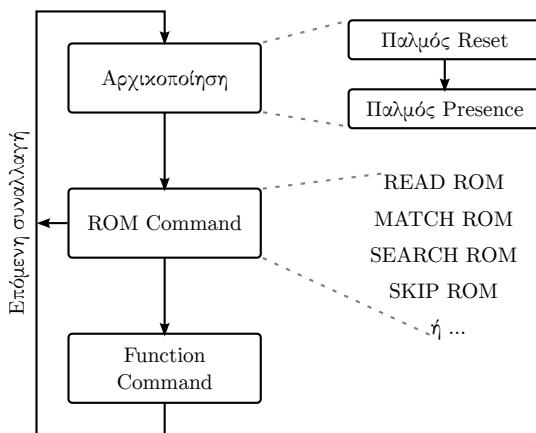
Σχήμα 3.5.1: Χρονοθυρίδες εγγραφής και ανάγνωσης στο δίαυλο 1-Wire.



Βασισμένο Figure 14. Read/Write Time Slot Timing Diagram. Στο Maxim Integrated. *DS18B20 Programmable Resolution 1-Wire® Digital Thermometer*. 22 Απρ. 2008. 27 σσ. url: <http://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html> (επίσκεψη 10/03/2014), σ. 16

Σχήμα 3.5.2: Κύκλος συναλλαγής στο δίαιρο 1-Wire.

Όλες οι συναλλαγές ξεκινούν με τους παλμούς αρχικοποίησης και ακολουθεί η επιλογή της επιλυμητής slave συσκευής και η αποστολή εντολής προς αυτήν. Το ακριβές σύνολο εντολών λειτουργίας (Function Command) εξαρτάται από την οικογένεια της εκάστοτε συσκευής (για παράδειγμα, λήψη θερμοκρασίας από αισθητήρα θερμοκρασίας).



όπως γίνεται ορατό και στο σχήμα, ο χρόνος κατά τον οποίο η γραμμή παραμένει σε λογικό 0 ως επενέργεια του master είναι όσο το δυνατό μικρότερος, ενώ η ανάγνωσή της γίνεται όσο το δυνατό κοντινότερα στη λήξη του διαστήματος των 15μs. Με αυτόν τον τρόπο, μειώνεται σημαντικά η πιθανότητα εσφαλμένης ανάγνωσης που οφείλεται σε μη σταθεροποιημένο σήμα.

Όλες οι χρονοθυρίδες, ανεξαρτήτως εάν πρόκειται για εγγραφή ή ανάγνωση, διαρκούν από 60 έως 120μs και ακολουθούνται από 1μs, τουλάχιστον, χρόνο ανάκτησης ( $T_{REC}$ ) κατά τον οποίο η γραμμή αφήνεται ελεύθερη (Maxim, 2008b, σσ. 15–16. Atmel, 2004, σ. 2).

### Παλμοί αρχικοποίησης

Η έναρξη της επικοινωνίας μεταξύ master και slave γίνεται με την αποστολή ενός παλμού Reset από το master και την απόχριση του/των slave με παλμό Presence (Maxim, 2008b, σ. 15). Ο παλμός Reset θέτει τη γραμμή σε λογικό 0 τουλάχιστον για 480μs (διάστημα 8 φορές μεγαλύτερο από τις χρονοθυρίδες ανάγνωσης/εγγραφής), κατόπιν η γραμμή απελευθερώνεται και εφόσον, εντός 60μs από τη στιγμή εκείνη, η γραμμή βρίσκεται σε λογικό 0, τότε υπάρχει τουλάχιστον μία συνδεδεμένη slave συσκευή στο δίαιρο (παλμός Presence) (Atmel, 2004, σ. 3).

### Συναλλαγές

Στο δίαιυλο 1-Wire, εκτός από τους παλμούς αρχικοποίησης και των χρονοθυρίδων ανάγνωσης/εγγραφής, ορίζονται ακολουθίες εντολών (δηλαδή, ανταλλαγή συγκεκριμένων τιμών Byte), αναφερόμενες ως συναλλαγές, για την επίτευξη οποιασδήποτε λειτουργίας (Maxim, 2008b, σ. 10). Το σχήμα 3.5.2 παρουσιάζει τις εργασίες μίας συναλλαγής. Η αρχικοποίηση έχει περιγραφεί, ενώ οι εντολές λειτουργίας (Function Command) εξαρτώνται από τα εκάστοτε είδη των συνδεδεμένων συσκευών.

**Εντολές ROM** Ο λόγος ύπαρξης των εντολών ROM αιτιολογείται από την ανάγκη για τον εντοπισμό και επιλογή μίας συσκευής slave εκ του συνόλου των συνδεδεμένων στο δίαιυλο. Προκειμένου να είναι αυτό δυνατό, αποδίδεται σε κάθε συσκευή ένα μοναδικό αναγνωριστικό των 64bit το οποίο περιέχει έναν κωδικό οικογενείας της συσκευής των 8bit, ένα σειριακό αριθμό των 48bit και κωδικό CRC (Cyclic Redundancy Check) των 8bit βάσει των προηγούμενων Byte (Maxim, 2008b, σ. 6). Οι κοινώς υποστηριζόμενες εντολές ROM είναι οι ακόλουθες:

**READ ROM [0x33]** Επιτρέπει την απευθείας ανάγνωση του αναγνωριστικού της μοναδικής slave συσκευή του διαιύλου (δεδομένου ότι είναι μόνο μία).

**MATCH ROM [0x55]** Για την αναγγελία του επιθυμητού αναγνωριστικού των 64bit από το master και την απόχριση μόνο από το slave που τον διαθέτει. Το συγκεκριμένο μπορεί να χρησιμοποιηθεί εάν, για παράδειγμα, είναι, εκ των προτέρων, καταχωρημένα όλα τα πιθανά αναγνωριστικά.

**SEARCH ROM [0xF0]** Χρησιμοποιείται για την ανάγνωση των αναγνωριστικών των συνδεδεμένων συσκευών, ένα προς ένα, εφόσον τα ακριβή στοιχεία τους είναι άγνωστα κατά την εκκίνηση του συστήματος.

**SKIP ROM [0xCC]** Παράληψη της επιλογής κάποιας συγκεκριμένης συσκευής επειδή πρόκειται να δούθει η ίδια εντολή σε όλες τις συσκευές ή επειδή είναι γνωστό ότι μόνο μία είναι συνδεδεμένη.

### ATmega328P και 1-Wire

Στο μικροελεγκτή ATmega328P, ο δίαιυλος 1-Wire είναι δυνατό να υποστηριχθεί εξ ολοκλήρου μέσω λογισμικού ή εν μέρει σε λογισμικό και εν μέρει σε υλικό (για παράδειγμα, USART), χωρίς, ωστόσο, να υπάρχει εγγενώς κάποιο κύκλωμα ειδικά για αυτόν τον σκοπό (Atmel, 2004, σ. 3). Στο πλαίσιο της υλοποίησης, επιλέγεται η πρώτη προσέγγιση (εξ ολοκλήρου σε λογισμικό).

Επιπλέον, δεδομένου ότι χρησιμοποιείται μόνο μία συσκευή (αισθητήρας θερμοκρασίας), επιλέγεται η μη υλοποίηση της διαδικασίας αναζήτησης slave συσκευής (Search ROM), καθώς η μία που χρησιμοποιείται είναι, εξαρχής, γνωστή.

### 3.5.2 Δίαυλος I<sup>2</sup>C (TWI)

Βασικά χαρακτηριστικά του διαύλου είναι η διασύνδεση των συσκευών με δύο γραφμές, τις SDA (Serial DAta) και SCL (Serial CLock), τη δυνατότητα οποιασδήποτε συσκευής να ξεκινήσει την επικοινωνία με κάποια άλλη (multi-master) επιλέγοντάς την μέσω μοναδικής διεύθυνσης που διαθέτουν όλες με συχνότητα του ρολογιού που αναγνωρίζεται από τις ίδιες τις συσκευές (δεδομένου ότι είναι εντός των επιτρεπτών τους ορίων), χωρίς να απαιτείται προηγούμενη ρύθμιση κάθε συσκευής ξεχωριστά (NXP, 2014, σσ. 3–4,6).

Σύμφωνα με την NXP (2014, σ. 6), προκειμένου κάποια συσκευή να επικοινωνήσει με κάποια άλλη, απαιτείται πρώτα να αποκτήσει τον έλεγχο του διαύλου (να γίνει master). Κατόπιν, αποστέλλει τη διεύθυνση του επιθυμητού slave (7bit) ακολουθούμενο από το bit πρόθεσης (λογικό 1 συνεπάγεται ανάγνωση από το slave) και λαμβάνει ένα bit επιβεβαίωσης (ACK) από το slave, σε περίπτωση επιτυχίας ενώ στη συνέχεια ακολουθούν τα Byte καθένα συνοδευόμενο από bit επιβεβαίωσης ή μη (NXP, 2014, σσ. 10,13).

Σε κατάσταση αδράνειας του διαύλου, οι γραφμές SDA και SCL ηρεμούν σε λογικό 1 μέσω αντιστατών pull-up (αντίστασης, συνήθως, μεταξύ 2–10kΩ) ενώ όλες οι συνδεδεμένες συσκευές βρίσκονται σε κατάσταση υψηλής εμπέδωσης (Phillips, 2003, σ. 13. NXP, 2014, σ. 8). Σε τυπική λειτουργία, η γραφμή SDA τίθεται (προετοιμάζεται) όταν η γραφμή SCL βρίσκεται σε λογικό 0. Τη στιγμή που η SCL ανέρχεται, η SDA θεωρείται ότι έχει σταθεροποιηθεί και οι συνδεδεμένες συσκευές διαβάζουν την τιμή της. Κατόπιν, η SCL επανέρχεται σε λογικό 0 ώστε η SDA να τεθεί εκ νέου (NXP, 2014, σ. 9).

Επιπλέον, ορίζονται δύο ειδικές περιπτώσεις που χρησιμοποιούνται για την εκκίνηση και τον τερματισμό της επικοινωνίας από το master – η αποστολή των bit START και STOP – τα οποία αποτελούν τη μεταβολή της SDA σε λογικό 0 και 1, αντιστοίχως, ενώ η γραφμή SCL βρίσκεται σε λογικό 1 (NXP, 2014, σ. 9). Μία υποπερίπτωση αυτών, είναι η αποστολή bit START από το master του διαύλου ώστε να πραγματοποιήσει νέα επικοινωνία (είτε με την ίδια είτε με άλλη συσκευή) χωρίς να απελευθερώσει το δίαυλο πρώτα το δίαυλο (NXP, 2014, σ. 13). Το bit αυτό αποκαλείται Sr (Repeated Start) και είναι ισοδύναμο του bit START με την εξαίρεση ότι στην περίπτωση που ο master λαμβάνει από το slave πρέπει να αποκριθεί με NACK προτού αποστέλλει το bit Sr (NXP, 2014, σσ. 13–14).

Ο δίαυλος I<sup>2</sup>C υποστηρίζει, επίσης, μηχανισμούς διαιτησίας (arbitration) και

αναγνώρισης συγχρούσεων (collision detection) που αξιοποιούνται σε υλοποιήσεις πολλαπλών συσκευών που γίνονται master του διαύλου. Στην περίπτωση αυτής της υλοποίησης, master του διαύλου μπορεί να είναι μόνο ο μικροελεγκτής και, συνεπώς, αποφεύγεται η περαιτέρω ανάλυσή τους.

Ο μικροελεγκτής της υλοποίησης (ATmega328P) διαθέτει κύκλωμα συμβατό με το I<sup>2</sup>C της Phillips υπό το όνομα TWI (Two-Wire Interface) και δια μέσω αυτού πραγματοποιείται η διασύνδεσή του με το ολοκληρωμένο (Atmel, 2013, σ. 209).

### 3.5.3 Δίαυλος SPI

Η διασύνδεση των συσκευών σε δίαυλο SPI (Serial Peripheral Interface) γίνεται μέσω τριών, κοινών για όλες τις συσκευές, γραμμών για την ανταλλαγή των δεδομένων – SCK (Serial Clock), MOSI (Master-Out Slave-In), MISO (Master-In Slave-Out) – και επιπρόσθετων γραμμών για την επιλογή κάθε slave συσκευής (SS – Slave Select) (Motorola, 2004, σσ. 15,24). Η master συσκευή είναι υπεύθυνη για την επιλογή της επιθυμητής, για επικοινωνία, slave συσκευής καθώς και για την παραγωγή του ρολογιού, ενώ τα δεδομένα αποστέλλονται ταυτόχρονα και προς τις δύο κατευθύνσεις (Motorola, 2004, σσ. 26–27).

Στο σχήμα 3.5.3 παρουσιάζονται τα σήματα επικοινωνίας μέσω του διαύλου SPI. Ο δίαυλος ρυθμίζεται για τη χρήση είτε λογικού 1 είτε 0 ως ενεργό bit των παλμών ρολογιού (πολικότητα – polarity) καθώς και σε ποια παρυφή των παλμών του ρολογιού πραγματοποιείται η δειγματοληψία από τις master και slave συσκευές (Motorola, 2004, σσ. 27–28). Στο παράδειγμα του σχήματος, η δειγματοληψία των γραμμών MOSI και MISO πραγματοποιείται κατά την μπροστινή παρυφή των παλμών του ρολογιού (για παράδειγμα, με πολικότητα CPOL = 0, στην ανερχόμενη). Αντιστοίχως, ο δίαυλος υποστηρίζει τη δειγματοληψία στην οπίσθια παρυφή· η ρύθμιση γίνεται μέσω της φάσης (Clock Phase). Οι επιλογές φάσης και πολικότητας ορίζουν τέσσερις πιθανές ρυθμίσεις λειτουργίας του διαύλου SPI, η εφαρμογής ποιας πρέπει να είναι εξαρχής γνωστή και προσυμφωνημένη μεταξύ master και slave, ενώ είναι δυνατό να εφαρμόζεται σε κάθε επικοινωνία και μία διαφορετική, σύμφωνα με τις απαιτήσεις της κάθε συσκευής (Motorola, 2004, σ. 27. Atmel, 2013, σ. 167).

Ο μικροελεγκτής ATmega238P υποστηρίζει εγγενώς το δίαυλο SPI καθώς διαθέτει κύκλωμα αφιερωμένο κύκλωμα. Υποστηρίζει και τις τέσσερις ρυθμίσεις λειτουργίας βάσει συνδυασμού πολικότητας-φάσης με δυνατότητα επιλογής της σειράς αποστολής των bit (πλέον ή λιγότερο σημαντικό bit) ενώ η συχνότητα του ρολογιού μπορεί να φτάνει μέχρι το μισό της συχνότητας του ρολογιού συστήματος. Οι επιλογές αυτές γίνονται δια μέσω του καταχωρητή SPCR (SPI Control Register) (Atmel, 2013, σ. 169). Παρότι υποστηρίζεται, επιλέγεται η μη αξιοπο-

Σχήμα 3.5.3: Χρονισμός δεδομένων SPI με φάση φολογιού CPHA = 0.

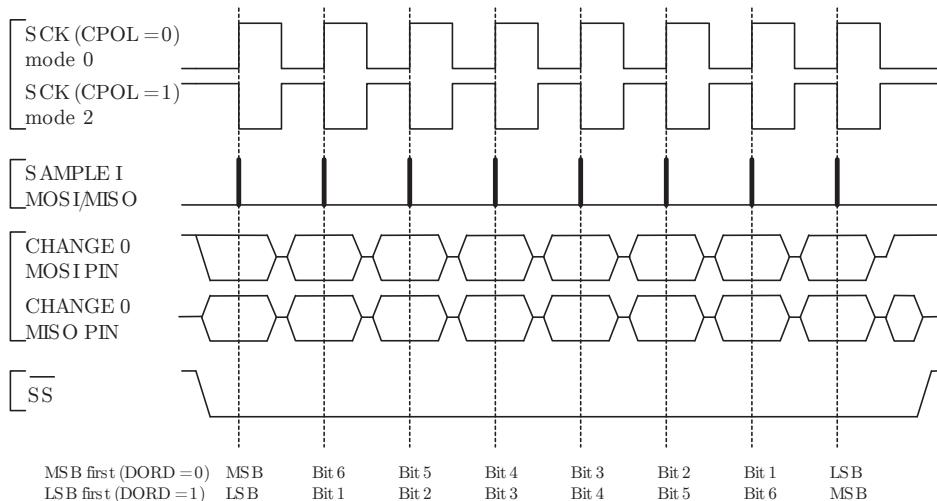


Figure 19-3. SPI Transfer Format with CPHA = 0. Στο Atmel Corporation. ATmega48A, ATmega48PA, ATmega88A, ATmega88PA, ATmega168A, ATmega168PA, ATmega328, ATmega328P datasheet. Εκδ. 1.7. 12 Φεβ. 2013. 660 σσ. url: <http://www.atmel.com/devices/ATMEGA328.aspx?tab=documents> (επίσκεψη 01/01/2014), σ. 168

ηση διακοπής ως αποτέλεσμα αποστολής/λήψης κάθε Byte, αλλά, αντιθέτως, η αναμονή μέχρι την ενεργοποίηση της ένδειξης SPIF (SPI Interrupt Flag) του καταχωρητή SPSR (SPI Status Register) από το υποκείμενο κύκλωμα (Atmel, 2013, σσ. 167,170).

Στο πλαίσιο της υλοποίησης, ο δίαυλος SPI χρησιμοποιείται για τη διασύνδεση με το ολοκληρωμένο δικτυωτής σύνδεσης (βλ. Το ολοκληρωμένο δικτύωσης W5100 σ. 105) και την εξωτερική μνήμη Flash, ενώ ως λειτουργία χρησιμοποιείται η SPI mode 0 (δηλαδή, CPOL = 0, CPHA = 0) καθώς υποστηρίζεται και από τα δύο ολοκληρωμένα.

## Κεφάλαιο 4

### Κατασκευή

Το κεφάλαιο ασχολείται με την επιλογή και συνδυασμό μηχανικών εξαρτημάτων προκειμένου να συντεθεί η φυσική υπόσταση της συσκευής (hardware), η οποία προορίζεται να ελέγχεται από το λογισμικό του μικροελεγκτή.

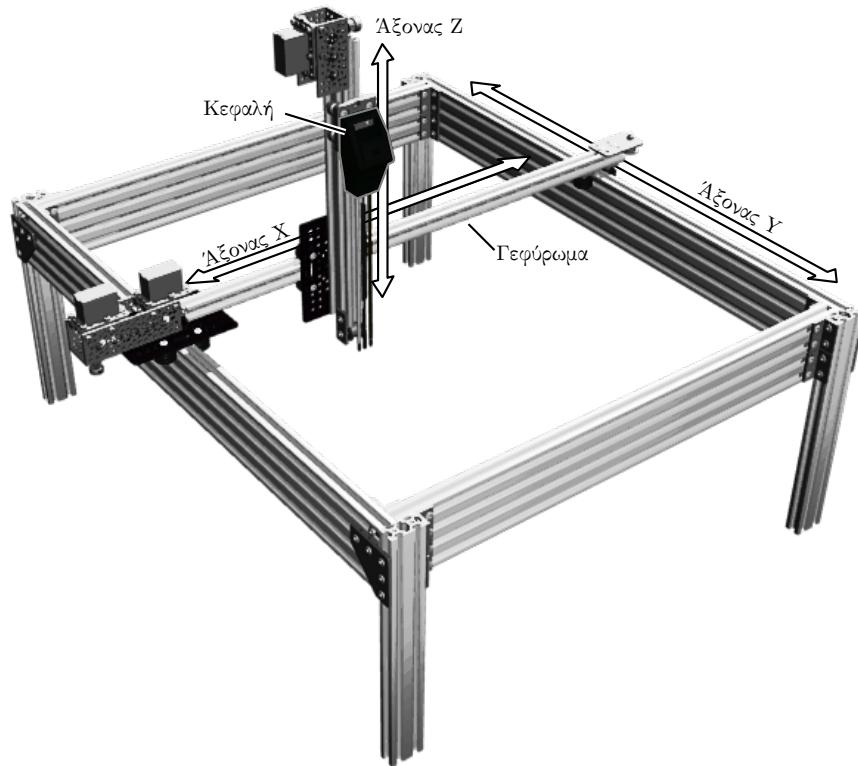
Το σημαντικότερο στοιχείο της συσκευής είναι η κεφαλή, η οποία πρόκειται για ένα όργανο που κινείται σε επίπεδο πάνω από το παρακολουθούμενο υλικό και κατακόρυφα προς αυτό εισχωρώντας το, ώστε τα αισθητήρια όργανα που φέρει να έρχονται σε επαφή με το υλικό και να πραγματοποιούν μετρήσεις των συνθηκών που επικρατούν σε κάθε σημείο.

Κρίνεται ότι ο συνδυασμός κινητής κεφαλής και αισθητήρων έχει το πλεονέκτημα χρήσης μικρού αριθμού αισθητήρων καθώς η κινητικότητά τους τους καθιστά ικανούς να καλύπτουν όλο το υλικό. Επίσης, σε ενδεχόμενη ανάγκη παρακολούθησης υλικού που καταλαμβάνει ένα μικρό μέρος της μέγιστης υποστηριζόμενης επιφάνειας, η οποιαδήποτε ρύθμιση αρκεί να γίνεται μέσα από το λογισμικό της συσκευής ώστε, απλώς, να περιορίζεται η επιφάνεια που καλύπτει. Ωστόσο, απαιτείται μηχανισμός που υποστηρίζει τη γραμμική κίνηση της κεφαλής στο χώρο καθώς και υποδομή που επιτρέπει τη στήριξη των σχετικών εξαρτημάτων.

Αρχικά, μελετώνται τα συστήματα CNC (Computer Numerical Control). εργαλειομηχανές (όπως τόρνος, γραναζοκόπτης, τρισδιάστατος εκτυπωτής) των οποίων ο χειρισμός γίνεται μέσω υπολογιστή παρέχοντας πολύ μεγαλύτερη ταχύτητα και ακρίβεια στην εργασία (Μηχανή CNC σ. 50). Ωστόσο, όπως αναφέρεται και στη σχετική ενότητα, η μελέτη τους γίνεται, όχι τόσο για την κατανόηση του τρόπου λειτουργίας τους, αλλά για τον εντοπισμό μίας κατάλληλης δόμησης της συσκευής που, όπως φαίνεται στο σχήμα 4.0.1, ακολουθεί διάταξη κινητού γεφυρώματος.

Τα στοιχειώδη δομικά στοιχεία (οδηγοί/ράγες, πλάκες συνένωσης) παρέχονται από σύστημα κατασκευής (construction framework) το οποίο επιλέγεται με γνώμονα την παροχή ευελιξίας κατά την ανάπτυξη του αρχετύπου ιδίως με ανασχε-

Σχήμα 4.0.1: Αναπαράσταση της συσκευής.



διασμό μερών της συσκευής καθώς η υλοποίησή της βρίσκεται ακόμα σε εξέλιξη. Τα βασικά στοιχεία του επιλεγμένου συστήματος κατασκευής, OpenBuilds, αναφέρονται στη σελίδα 54.

Στην πορεία, ακολουθούν τα βασικά μέρη που απαρτίζουν τη συσκευής και πώς αυτά συντίθενται από τα διαθέσιμα δομικά στοιχεία OpenBuilds. Αρχικά, αναφέρεται η βάση της συσκευής, η οποία πρόκειται για το πλαίσιο πάνω στο οποίο πραγματοποιείται όλη η κίνηση (σ. 58). Ακολουθούν οι οδηγοί κίνησης (άξονες X, Y και Z) (σ. 59) και, τελικά, η τοποθέτηση των κινητήρων (σ. 63).

## 4.1 Μηχανή CNC

Μηχανή αριθμητικού ελέγχου (Numerical Control – NC) είναι μία μηχανή η οποία δέχεται κωδικοποιημένες οδηγίες για την εκτέλεση των επιθυμητών ενεργειών και, κατά βάση, αναφέρεται στον έλεγχο των εξαρτημάτων της συσκευής παρά σε συγκεκριμένου τύπου μηχανή (Seames, 2001. Albert, 2011).

Αρχικά, οι εντολές δίνονται στη μηχανή μέσω διάτρητων καρτών ή χαρτο-

ταινιών, ωστόσο, η εξέλιξη των μικροϋπολογιστών επέτρεψε τη χρήση τους για την εκτέλεση του προγράμματος (Computer Numerical Control - CNC) (Seames, 2001).

Δεδομένου του ορισμού μία μηχανής CNC, η συσκευή είναι αδύνατο να θεωρηθεί ως τέτοια. Βασικός λόγος είναι ότι η συσκευή αυτοματοποιεί μία διαδικασία δεχόμενη κάποιες παραμέτρους που ρυθμίζουν ελαφρώς ορισμένες λειτουργίες της χωρίς, ωστόσο, να απαιτείται η παροχή ενός συνόλου οδηγιών οι οποίες ελέγχουν τα μέρη της για την πραγματοποίηση της επιθυμητής εργασίας, όπως δηλαδή, απαιτείται από μηχανές CNC. Ωστόσο, κρίνεται χρήσιμη η γνωριμία με τέτοια συστήματα καθώς, ακολουθώντας μία εξελικτική πορεία από το 1952, έχει συσσωρευτεί εμπειρία η οποία μπορεί να φανεί χρήσιμη για τις ανάγκες της συσκευής.

Ένα από τα ζητήματα που έχουν κληθεί να αντιμετωπίσουν είναι η κίνηση των εξαρτημάτων τους στο χώρου. Έχουν αναπτυχθεί διάφορες για το σκοπό αυτό οι οποίες μελετώνται για την ανάδειξη κάποιας κατάλληλης για την υλοποίηση.

#### 4.1.1 Διατάξεις CNC

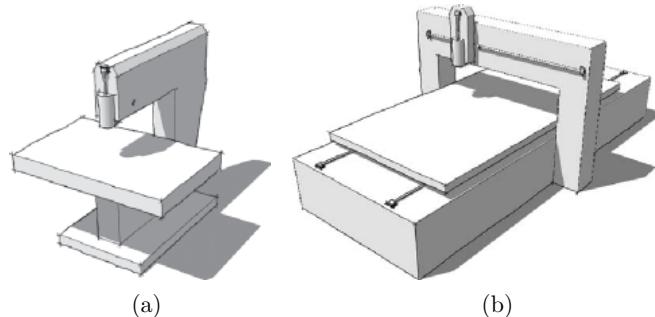
Η τράπεζα ενός CNC είναι η επιφάνεια πάνω στη οποία εναποτίθεται το προς επεξεργασία υλικό του συστήματος η οποία είναι δυνατό να κινείται προκειμένου να εξυπηρετεί κίνηση σε κάποιον όξονα ή να είναι πλήρως σταθερή. Βάσει αυτού του χαρακτηριστικού, οι διατάξεις CNC διαχρίνονται σε κινητής και σταθερής τραπέζης.

Πέραν των διατάξεων που αναφέρονται παρακάτω, υφίστανται κι άλλες που βρίσκουν εφαρμογή στην επεξεργασία μεγάλων τρισδιάστατων αντικειμένων, όπως διατάξεις 5 διαστάσεων ή βιομηχανικά ρομπότ, υποστηρίζοντας περιστροφική κίνηση γύρω από άξονες επιπροσθέτως της ευθύγραμμης κίνησης σε αυτούς, οι οποίες ξεπερνούν δραματικά τις απαιτήσεις της υλοποίησης και, συνεπώς, αποκλείονται από αυτήν.

##### Κινητή τράπεζα

Αναφέρονται δύο διατάξεις που χρησιμοποιούν κινητή τράπεζα. Σε κάθε περίπτωση, η κίνηση στον κατακόρυφο προς την τράπεζα όξονα, Z, εκτελείται από σταθερό σημείο. Στην πρώτη διάταξη, η τράπεζα κινείται τόσο στον όξονα X όσο και στον όξονα Y, με την κίνηση στον όξονα Z να εκτελείται από σταθερή δοκό που εκτείνεται πάνω από την τράπεζα, στερεωμένη στη βάση μέσω κατακόρυφης στήλης (σχήμα ??α) (Albert, 2011, σ. 69). Η τράπεζα αυτής της διάταξης περιορίζεται σε, σχετικά, μικρές διαστάσεις καθώς, παράλληλα με αυτές, αυξάνει και το μήκος της δοκού και οι απαιτήσεις για τη στερέωσή της.

Σχήμα 4.1.1: Διατάξεις κινητής τραπέζης.



(α'): X-Y tables have evolved from pin routers. Στο Alain Albert. *Understanding CNC Routers*. FPIInnovations, 2011. 105 σσ.

(β'): Moving Table Router. Στο Alain Albert. *Understanding CNC Routers*. FPIInnovations, 2011. 105 σσ.

Σε εναλλακτική διάταξη, η κίνηση στο επίπεδο X-Y διαμοιράζεται μεταξύ της τραπέζης και μίας κεφαλής η οποία κινείται στον έναν εκ των δύο αξόνων πάνω σε γεφύρωμα που διατρέχει όλο το πλάτος της συσκευής (σχήμα ??β) (Albert, 2011, σ. 70).

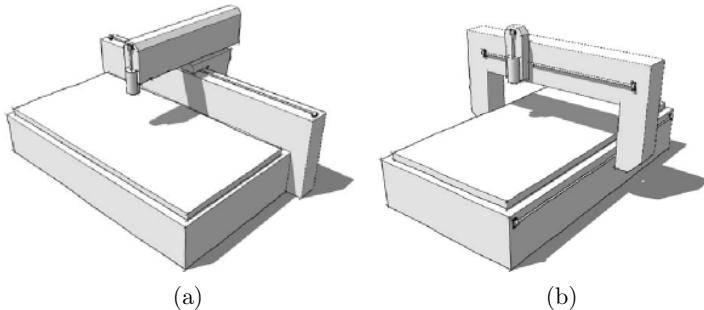
Οι διατάξεις κινητής τραπέζης έχουν το μειονέκτημα ότι η κίνηση της τραπέζης επιβαρύνεται από το υπό παρακολούθηση υλικό θέτοντας περιορισμούς στο μέγιστο επιτρεπτό βάρος του φορτίου. Ένα δεύτερο μειονέκτημα είναι ότι η βάση της συσκευής πρέπει να είναι αρκετά μεγαλύτερη από την τράπεζα μόνο και μόνο για την εξυπηρέτηση της κίνησης με αποτέλεσμα να δεσμεύεται επιπρόσθετος χώρος χωρίς αυτός να αξιοποιείται ουσιαστικά για τις ανάγκες της υποβοηθούμενης διαδικασίας.

### Σταθερή τράπεζα

Σε μία πρώτη διάταξη σταθερής τραπέζης χρησιμοποιείται κινητή στήλη στήριξης μίας, επίσης, κινητής δοκού, έκαστη κινούμενη σε διαφορετικό άξονα (σχήμα 4.1.2α) (Albert, 2011, σ. 70). Άμεση πρόκληση αυτής της διάταξης, η οποία απορρέει από το ότι η δοκός στηρίζεται μόνο σε μία πλευρά, σχετίζεται με τη σταθερότητα της δοκού ιδίως με την επιμήκυνση της για την κάλυψη μεγαλύτερης επιφάνειας.

Εναλλακτική διάταξη μοιάζει με τη δεύτερη διάταξη κινούμενης τραπέζης που έχει αναφερθεί με τη διαφορά ότι αντί για την τράπεζα, κινείται το γεφύρωμα (σχήμα 4.1.2β) (Albert, 2011, σ. 71). Ένα ενδεχόμενο μειονέκτημα αυτής της περίπτωσης είναι η ανάγκη για ύπαρξη εξαρτημάτων κίνησης σε κάθε άκρο του γεφυρώματος.

Σχήμα 4.1.2: Διατάξεις σταθερής τραπέζης.



(α'): *Cantilevered Arm Router*. Στο Alain Albert. *Understanding CNC Routers*. FPIInnovations, 2011. 105 σσ.

(β'): *Moving Gantry Router*. Στο Alain Albert. *Understanding CNC Routers*. FPIInnovations, 2011. 105 σσ.

#### 4.1.2 Υπόδειγμα κατασκευής

Μεταξύ των δύο κατηγοριών διατάξεων που έχουν αναφερθεί, προτιμώνται διατάξεις σταθερής τραπέζης κι αυτό επειδή, αφενός, αξιοποιούν καλύτερα το χώρο που τους αφιερώνεται, αφετέρου επηρεάζουν και επηρεάζονται λιγότερο από τα χαρακτηριστικά του δοχείου.

Σε σχέση με το δεύτερο, κρίνεται αναπόφευκτη η εξάρτηση των διαστάσεων του δοχείου από τις διαστάσεις της συσκευής, δεδομένου ότι η συσκευή το πλαισιώνει και κινείται γύρω από αυτό, ανεξαρτήτως της επιλεγμένης διάταξης. Ωστόσο, η μάζα του φορτίου, η οποία εξαρτάται από τη μάζα των περιεχόμενων υλικών καθώς και την απορροφητικότητα τους και την εκάστοτε υγρασία που επικρατεί στο δοχείο, χρήζει μελέτης μόνο στην περίπτωση των διατάξεων κινητής τραπέζης. Επιλέγοντας διατάξεις που διατηρούν μονίμως σταθερή την τράπεζα, εξαλείφονται ενδεχόμενοι περιορισμοί στο μέγιστο επιτρεπτό φορτίο.

Επόμενο στοιχείο που μελετάται είναι η αναγκαιότητα ύπαρξης τραπέζης. Δεδομένου ότι στο δοχείο επικρατούν συνθήκες υψηλής υγρασίας και η κατασκευή αποτελείται κυρίως από μέταλλο, κρίνεται σκόπιμο τα δύο να είναι πλήρως ανεξάρτητα. Παράδειγμα αποτελεί η αποστράγγιση, η οποία αν παρέχεται από το δοχείο είναι προτιμότερο να μην έρχεται σε επαφή με τη συσκευή. Με αυτόν τον τρόπο, αποφεύγεται η οξείδωση μερών της συσκευής ενώ παράλληλα επιτρέπεται η ελεύθερη επιλογή, διαμόρφωση και αντικατάσταση του δοχείου χωρίς να επηρεάζεται η ίδια η συσκευή. Η ανεξαρτησία ορισμένων ιδιοτήτων του δοχείου σε σχέση με τις δυνατότητες της συσκευής σε συνδυασμό με το ότι μπορεί να μετακινείται ελεύθερα χωρίς να απαιτείται πρόσδεση με τη βάση, αποτελούν τους βασικούς λόγους για την περαιτέρω απάλειψη της τραπέζης.

Σε σχέση με τις δύο προαναφερθείσες διατάξεις σταθερής τραπέζης, επιλέγεται αυτή του κινητού γεφυρώματος επειδή παρέχει σταθερότητα με περισσότερη ευκολία.

## 4.2 Σύστημα κατασκευής

Κρίνεται απαραίτητη η εύρεση και αξιοποίηση ενός συστήματος κατασκευής αρχετύπων που επιταχύνει τη διαδικασία κατασκευής παρέχοντας ευελιξία στην τροποποίηση της υλοποίησης καθώς αυτή αναπτύσσεται και προκύπτουν νέες απαιτήσεις ή και προβλήματα, η οποία, ενδεχομένως, παρέχει λύσεις σε συχνές ανάγκες στο πλαίσιο κατασκευής.

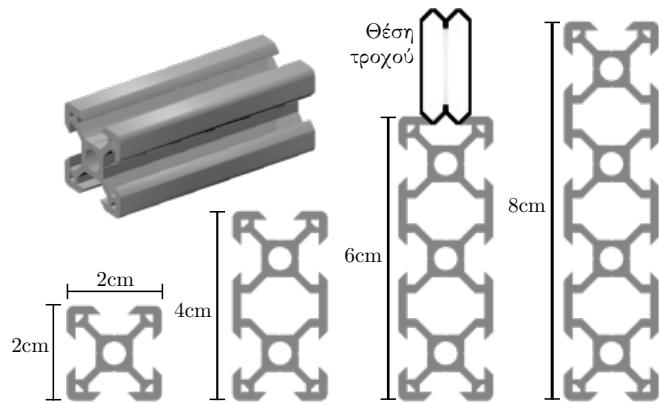
Εντοπίστηκαν αρκετά συστήματα (ουσιαστικά το 80/20 και παραλλαγές αυτού όπως Misumi, MakerBeam) τα οποία χρησιμοποιούν ράβδους εξωθημένου αλουμινίου ως δομικά στοιχεία τα οποία συνδυάζονται μεταξύ τους μέσω προσαρτημάτων για την κατασκευή του σκελετού της κατασκευής.

Η εξώθηση αλουμινίου είναι μία τεχνολογία πλαστικής, δηλαδή μόνιμης, παραμόρφωσης όπου αλουμίνιο εξωθείται να διέλθει μέσω ενός ανοίγματος συγκεκριμένου σχήματος για την δημιουργία ράβδων αντίστοιχης διατομής, και τα προϊόντα της βρίσκουν εφαρμογή σε διάφορους τομείς όπως στην αρχιτεκτονική, τη βιομηχανία αυτοκινήτων και την παραγωγή μικρών μηχανικών και δομικών στοιχείων (Saha, 2000).

Το θετικό αυτής της λύσης είναι ότι χρησιμοποιούνται ξεχωριστά προσαρτήματα για τη στερέωση των ράβδων και όχι μόνιμη οξυγονοκόλληση επιτρέποντας την ανά πάσα στιγμή αναδιαμόρφωση των συνδέσεων ενόψει νέων απαιτήσεων ή προβλημάτων και, λόγω της μορφής τους, επιτρέπουν την ενασχόληση με μεμονωμένα τμήματα με την περαιτέρω σύνθεσή τους στην τελική κατασκευή. Για την εξυπηρέτηση ευθύγραμμης κίνησης, οι ράβδοι χρησιμοποιούνται σε συνδυασμό με τροχοφόρες διατάξεις ή διατάξεις που φέρουν ένσφαιρους τριβείς των ιδίων ή διαφορετικών συστημάτων.

Επιπροσθέτως αυτών, εντοπίστηκαν συστήματα που αφοισώνονται εξ ολοκλήρου στη γραμμική κίνηση, όπως Thomson Linear, Rexroth Bosch Group, NSK Linear. Μολονότι τα εν λόγω συστήματα, ενδεχομένως, χρησιμοποιούνται κατά κόρον σε βιομηχανικές εφαρμογές, οι απαιτήσεις αυτής της εφαρμογής είναι αρκετά διαφορετικές. Ενδιαφέρει περισσότερο η ευελιξία κατά την υλοποίηση (όπως η διυνατότητα χρήσης οποιασδήποτε δοκού ως οδηγό κίνησης), η ευκολία αναπροσαρμογής των στοιχείων (για παράδειγμα, χρήση αλουμινίου αντί του πιο σκληρού χάλυβα), και οι χαμηλές απαιτήσεις συντήρησης, όπως ελάχιστη ή καθόλου ανάγκη για λίπανση εξαρτημάτων (για παράδειγμα, για την επιμήκυνση της διάρκειας ζωής

Σχήμα 4.2.1: Τα τέσσερα μεγέθη οδηγών V-Slot.



Βασισμένο *VSlot Linear Rail* (2014). url: <http://openbuildspartstore.com/v-slot-20-x-20mm/> (επίσκεψη 13/04/2014). Mark Carew

τους), και όλα αυτά, ακόμα και εις βάρους της συνολικής απόδοσης της κατασκευής ή ενός λιγότερου κομφού αποτελέσματος.

Το επιλεγμένο σύστημα διαθέτει όλα τα προαναφερθέντα χαρακτηριστικά. Ένας επιπρόσθετος λόγος για την επιλογή αυτού έναντι αντίστοιχων συστημάτων είναι ότι παρέχει μία ποικίλη συλλογή δομικών στοιχείων χωρίς, ωστόσο, να είναι αχανής, ικανή να καλύψει πληθώρα αναγκών. Επιπλέον, υπερτερεί στο κομμάτι του αναμενόμενου χρόνου ζωής των τροχών, ενός αναμφισβήτητα σημαντικού δομικού στοιχείου αυτών των συστημάτων. Τέλος, τα στοιχεία παρέχονται ως ανοικτό υλικό με ελεύθερη τη λήψη αρχείων για την ενσωμάτωσή τους σε εφαρμογές σχεδίασης 3-Δ.

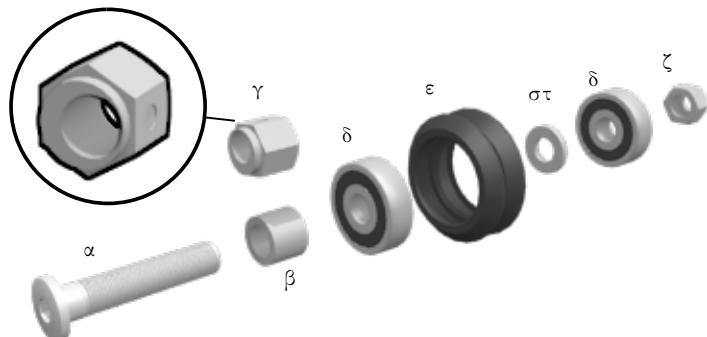
Στη συνέχεια παρουσιάζεται ο τρόπος ενσωμάτωσης δομικών στοιχείων του συστήματος OpenBuilds για τις ανάγκες της υλοποίησης.

### 4.2.1 Οδηγοί

Το βασικότερο δομικό στοιχείο του συστήματος είναι το VSlot, εφεξής οδηγός, το οποίο χρησιμοποιείται τόσο για την κατασκευή του σκελετού όσο και για γραμμική κίνηση, επιτρέποντας οποιοδήποτε τμήμα της κατασκευής να χρησιμοποιηθεί ως φορέας κινητών φορτίων.

Οι οδηγοί διαθέτουν αυλακώσεις οι οποίες, επικλινείς στο ανώτερο τμήμα, χρησιμοποιούνται ως διάδρομοι τροχών και ως υποδοχές μάντα, καλωδίων ή άλλων συνδετικών εξαρτημάτων στο κατώτερο τμήμα. Στο σχήμα 4.2.1 παρατίθεται η μορφή των τεσσάρων διαθέσιμων μεγεθών οδηγών. Κατασκευάζονται σε μήκος 1 και 1.5m από χράμα αλουμινίου 6063-T5 – μέταλλο μαλακό – το οποίο επιτρέπει

Σχήμα 4.2.2: Μέλη που αποτελούν τον τροχό.



την εύκολη αναπροσαρμογή του μήκους τους.

#### 4.2.2 Τροχοί

Συντίθενται από ανεξάρτητα μέρη, με αυτόν τον τρόπο επιτρέποντας την προσαρμογή τους στις απαιτήσεις κάθε περίπτωσης, την αντικατάσταση κάποιου πιλανού ελαττωματικού μέρους αντί ολόκληρου του τροχού καθώς και πιθανές μελλοντικές επεκτάσεις της κατασκευής τους. Το επίσωτρο των τροχών διαθέτει, σαφώς, συμβατή μορφή με τις αυλακώσεις των οδηγών και εφάπτεται στο ανώτερο τμήμα των αυλακώσεων δίχως να εισέρχεται πλήρως μέσα σε αυτές ώστε να αποφεύγονται φυσιορές στον τροχό κατά την κίνησή του από την επαφή του με τα τοιχώματα.

Στο σχήμα 4.2.2 παρατίθενται τα μέρη που συνθέτουν έναν υποδειγματικό τροχό, και, συνοπτικά περιγράφονται παρακάτω:

**Βίδα M5 (α)** ως άξονας περιστροφής και για την στερέωση του τροχού.

**Διαχωριστικό (β)** για την κάλυψη της απόστασης μέχρι την αυλάκωση.

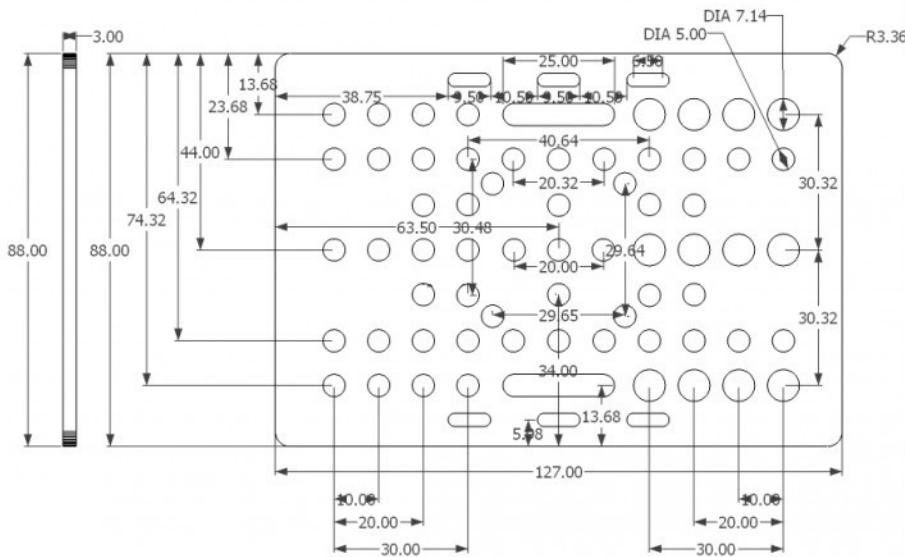
**Έκκεντρο διαχωριστικό (γ)** ως εναλλακτικό του απλού διαχωριστικού. Η χρήση αυτού αντί του απλού διαχωριστικού περιγράφεται σε επόμενη παράγραφο.

**Ένσφαιροι τριβείς (δ)** για την μείωση των τριβών μεταξύ τροχού και άξονα περιστροφής.

**Επίσωτρο (ε)** για την επαφή με την αυλάκωση.

**Δακτύλιος (στ)** ενδιάμεσος των τριβέων, για την αποφυγή συμπλοκής τους.

Σχήμα 4.2.3: Πλάκα γεφυρώματος (μονάδες σε mm).



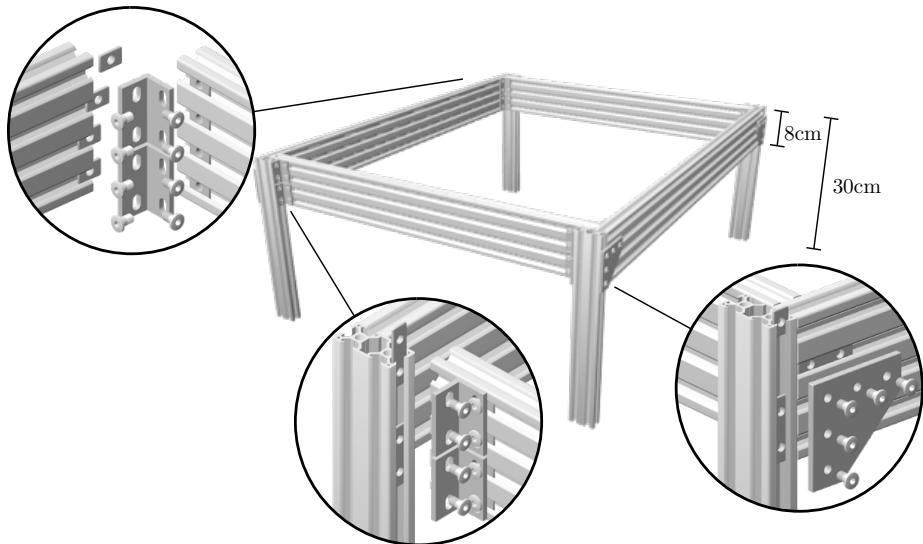
VSlot Universal Plate (2014). url: <http://openbuildspartstore.com/v-slot-gantry-plate/> (επίσκεψη 13/04/2014). Mark Carew

**Αντιπερικόχλιο (ζ)** για τη συγκράτηση του τροχού στον άξονα.

Οι τροχοί στερεώνονται απευθείας στις αυλακώσεις κάποιου οδηγού δίνοντάς του τη δυνατότητα κύλισης ή σε ξεχωριστή διάτρητη πλάκα η οποία μπορεί να φριλοξενήσει και πληρώρα άλλων εξαρτημάτων. Η βασικότερη πλάκα είναι η Universal Gantry Plate, εφεξής πλάκα γεφυρώματος, το αρχέτυπο της οποίας απεικονίζεται στο σχήμα 4.2.3. Οι περισσότερες κυκλικές οπές της είναι διαμέτρου 5mm ενώ οι επιμήκεις μπορούν να χρησιμοποιηθούν για την πρόσδεση ιμάντων.

Ορισμένες οπές είναι διαμέτρου 7.14mm και βρίσκονται συγκεντρωμένες στο δεξί μισό του σχήματος. Οι συγκεκριμένες χρησιμοποιούνται σε συνδυασμό με έκκεντρα διαχωριστικά έναντι απλών (σχήμα 4.2.2γ) των οποίων η προεξοχή εισέρχεται στην οπή. Περιστρέφοντας τους, μεταβάλλεται η απόσταση των αντίστοιχων τροχών από τον οδηγό έως ότου εφάπτονται ερμητικά με την αυλάκωση ώστε να αποφεύγονται οι κραδασμοί κατά την κίνηση της πλάκας. Τοποθετώντας τροχούς σε κατάλληλα ζεύγη οπών 5 και 7.14mm δημιουργείται διάκενο στο οποίο εισέρχεται οποιοδήποτε από τα τέσσερα μεγέθη οδηγών.

Σχήμα 4.3.1: Η βάση της συσκευής.



### 4.3 Βάση

Η βάση της συσκευής αποτελείται από πλαίσιο κατασκευασμένο από παραλληλεπίπεδους οδηγούς, αναρτημένο σε τέσσερις γωνιακούς οδηγούς στήριξης. Η συγκράτηση των οδηγών επιτυγχάνεται με χρήση γωνιακών προσαρτημάτων, κοχλιών M5 και περικοχλίων ένθετων στις αυλακώσεις των οδηγών (σχήμα 4.3.1). Όπως γίνεται αντιληπτό, στοιχεία όπως κοχλίες, περικόχλια, αντιπερικόχλια, δακτύλιοι κ.ο.κ. είναι απαραίτητα σε κάθε στάδιο της κατασκευής. Ωστόσο, σημειώνεται ότι αποφεύγεται η τόσο λεπτομερής ανάδειξη των επιμέρους αυτών στοιχείων, εφόσον κρίνεται ότι στερούνται ουσιαστικού ενδιαφέροντος.

Στον κενό χώρο που σχηματίζεται κάτω από το πλαίσιο, τοποθετείται το δοχείο παρακολούθησης το οποίο και τον καλύπτει χωρίς, ωστόσο, να εισέρχεται στο πλαίσιο. Ο χώρος του πλαισίου αφιερώνεται, εξ ολοκλήρου, στην κίνηση των οργάνων της συσκευής και είναι απαραίτητο να παραμένει ελεύθερος από εμπόδια. Αυτός είναι και ο λόγος που έχουν επιλεγεί για το πλαίσιο οι τόσο πλατιοί οδηγοί των 8cm· ώστε να καλύπτονται επαρκώς τα κινητά του όργανα καθώς και να παρέχεται ένα εύληπτο όριο για το ύψος του δοχείου. Περισσότερες λεπτομέρειες σχετικά με τα κινητά όργανα που προστατεύονται από το πλαίσιο δίνονται στην ενότητα Άξονας Z (σ. 61).

Το ύψος της βάσης και, συνεπώς, του δοχείου, περιορίζεται από το μέγεθος των ίδιων των αισθητήριων οργάνων καθώς όλα ήταν άσκοπο έως και επιβαρυτικό για την εξαγωγή έγκυρων μετρήσεων, εάν υπάρχει βάθος στο δοχείο το

οποίο παραμένει απροσπέλαστο από τους αισθητήρες και αδύνατο να καταμετρηθεί. Δεδομένου ότι οι επιλεγμένοι αισθητήρες υγρασίας και οξύτητας έχουν μήκος περίπου 21cm, για τη στήριξη του πλαισίου επιλέγονται οδηγοί 30cm, 8cm των οποίων χρησιμοποιούνται για την προσάρτησή τους στο πλαίσιο (σχήμα 4.3.1).

Οι άλλες δύο διαστάσεις είναι σχεδόν ανεξάρτητες με τους μοναδικούς περιορισμούς να τίθενται σχεδόν κατά αποκλειστικότητα από τις δυνατότητες του μικροεπεξεργαστή καθώς και από πρακτικούς λόγους όπως το μέγεθος της επιφάνειας προς διαχείριση. Ωστόσο, επειδή έχει γίνει ήδη αναφορά στο μέγεθος του δοχείου σε σχέση με τη βάση, σημειώνεται ότι η υπό παρακολούθηση περιοχή είναι ελαφρώς μικρότερη από τις πραγματικές εσωτερικές διαστάσεις της βάσης εξαιτίας ενός περιθωρίου μερικών εκατοστών – μίας περιοχής απροσπέλαστης από τα αισθητήρια όργανα. Οι λόγοι ύπαρξης του περιθωρίου γίνονται αντιληπτοί στις επόμενες ενότητες.

## 4.4 Άξονες κίνησης

### 4.4.1 Άξονας Y

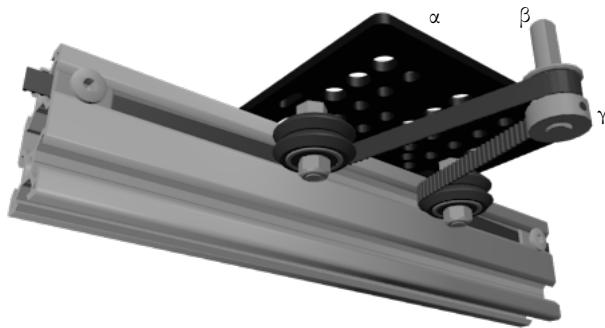
Τροχοφόρος πλάκα τοποθετείται παράλληλα προς το έδαφος με τους τροχούς της στην κορυφαία αυλάκωση ενός περιμετρικού οδηγού της βάσης. Στο σχήμα 4.4.1 απεικονίζεται η βασική διάταξη των εξαρτημάτων σε μία ενδεικτική απλοποιημένη υλοποίηση. Στην εξωτερική πλευρά της πλάκας (α) βρίσκεται κινητήρας από τον οποίο εκτείνεται ράβδος (β) που φέρει την κινητήρια τροχαλία (γ) (ο κινητήρας στερεώνεται πάνω στην πλάκα, ωστόσο, έχει αποκρυφτεί από το συγκεκριμένο σχήμα). Τραπεζοειδής ιμάντας τεντωμένος κατά μήκος της εξωτερικής αυλάκωσης του οδηγού και στερεωμένος στα όχρα του, αξιοποιεί τους τροχούς ως ελεύθερες τροχαλίες ώστε να αυξάνεται το τόξο επαφής με την κινητήρια τροχαλία (γ), παρέχοντας μεγαλύτερη σταθερότητα κατά την κίνηση.

### 4.4.2 Άξονας X

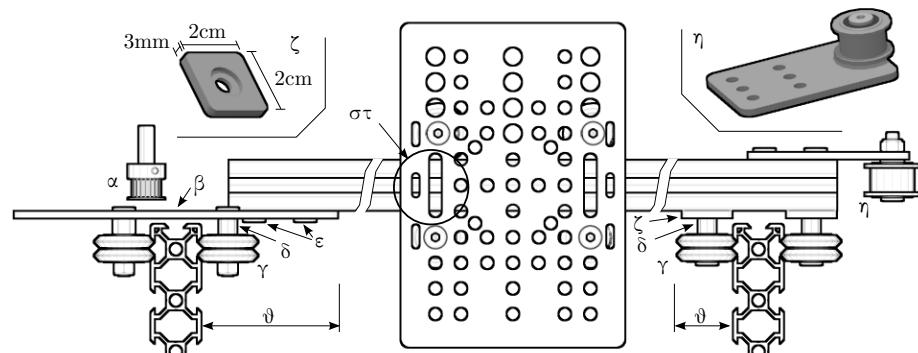
Στην ίδια πλάκα, στερεώνεται το ένα όχρο του γεφυρώματος, ένας νέου οδηγού, ο οποίος ενώνει αυτή με την απέναντι πλευρά της βάσης, πάνω στο οποίο εκτελείται η κίνηση στον άξονα X. Ωστόσο, επιλέγεται ελαφρώς διαφορετική προσέγγιση από αυτήν που χρησιμοποιήθηκε για τον άξονα Y. Βασικό λόγο αποτελεί η τοποθέτηση του κινητήρα και αυτού του άξονα στην ίδια πλάκα με αυτήν του κινητήρα του άξονα Y, προκειμένου η καλωδίωση του να εκτείνεται προς ένα κινητό σημείο και όχι, δυνητικά, όλου του πλάτους της βάσης. Επίσης, με αυτόν τον τρόπο αξιοποιείται μεγαλύτερο μέρος της επιφάνειας της πλάκας γεφυρώματος.

Σχήμα 4.4.1: Κινητή τροχαλία και ιμάντας.

Διάταξη τροχαλίας-ιμάντα για την κίνηση στον άξονα Y. Ο κινητήρας έχει αποκλειστεί από την απεικόνιση. Ωστόσο, νοείται ότι βρίσκεται συζευγμένος με τη ράβδο (β) στο ελεύθερο άκρο της.



Σχήμα 4.4.2: Συστατικά μέρη γεφυρώματος.



Στο σχήμα 4.4.2 παρουσιάζεται η υλοποίηση του άξονα X. Ο κινητήρας X τοποθετείται κοντά στο κέντρο της πλάκας γεφυρώματος (β) με τη βάση της τροχαλίας (α) παράλληλα προς την πλάκα και ελαφρώς υψηλότερα της ώστε ο ιμάντας να ευθυγραμμίζεται με αυλάκωση του γεφυρώματος. Η πλάκα φορτίου του άξονα X τοποθετείται κάθετα ως προς το δάπεδο και ο ιμάντας προσδένεται στις σχετικές πλαϊνές της οπές. Στην πλευρά της πλάκας πλησιέστερη της τροχαλίας (στ), ο ιμάντας προσδένεται απευθείας, ενώ στην άλλη, εφόσον διανύσει το μήκος του γεφυρώματος και αναστραφεί σε ελεύθερη τροχαλία (η) στο άλλο άκρο του.

Στο άκρο της ελεύθερης τροχαλίας, επιλέγεται διαφορετικός τρόπος στερέωσης και ανύψωσης του οδηγού από τη βάση. Ουσιαστικά, απαλείφεται η πλάκα γεφυρώματος και χρησιμοποιούνται δύο τροχοί αντί τεσσάρων, οι οποίοι στερεώνονται απευθείας στον οδηγό. Ο βασικότερος λόγος είναι η αξιοποίηση μεγαλύτερου μήκους του οδηγού, καθώς μία πλάκα γεφυρώματος, συνολικού μήκους 12.7cm,

επιφέρει μεγαλύτερο περιθώριο (θ) εσωτερικά της βάσης από ότι ένας τροχός. Η επιλογή αυτή ενδυναμώνεται περαιτέρω από το γεγονός ότι μία πλάκα στο σημείο αυτό θα χρησιμοποιούταν μόνο για τους τροχούς και την ελεύθερη τροχαλία.

Οστόσο, παραμένει βασική προϋπόθεση η χρήση κάποιου εξαρτήματος στη θέση της πλάκας που έχει το ίδιο πάχος, ώστε να διατηρείται οριζόντιος ο οδηγός. Το μικρότερο εξάρτημα που εντοπίστηκε είναι το πώμα (ζ) το οποίο, τυπικά, προορίζεται για την κάλυψη των άκρων των οδηγών. Η ορατή έδρα του στο σχήμα, η οποία διαθέτει μία κοιλότητα, εφάπτεται στον οδηγό. Η άλλη έδρα είναι πλήρως επίπεδη και σε αυτήν ακουμπά το διαχωριστικό. Επομένως, δεδομένου ότι χρησιμοποιούνται διαχωριστικά (δ) και επίσωτρα (γ) ίδιου ύψους, ο οδηγός διατηρείται οριζόντιος.

Η ελεύθερη τροχαλία δημιουργείται με επίσωτρο που παρέχει το σύστημα κατασκευής με τρόπο αντίστοιχο αυτού της σύνθεσης των τροχών. Συνεπώς, ελεύθερες τροχαλίες μπορούν να τοποθετηθούν απευθείας σε αυλακώσεις οδηγών ή σε πλάκες γεφυρώματος. Επίσης, παρέχεται και μία διαφορετική μικρότερη πλάκα με λιγότερες οπές ειδικά για αυτόν το σκοπό (απεικονιζόμενη στο σχήμα 4.4.2η).

Απευθείας προσάρτηση της ελεύθερης τροχαλίας σε αυλάκωση του οδηγού αποκλείεται σε αυτήν την περίπτωση, καθώς οι διαιθέσιμες αυλακώσεις καθιστούν αδύνατη την άμεση επικοινωνία της με την κινητήρια τροχαλία. Εφόσον έχει ήδη αποκλειστεί η χρήση πλάκας γεφυρώματος για την αποφυγή άσκοπης σπατάλης χώρου, επιλέγεται η προσάρτηση της ελεύθερης τροχαλίας στην ειδική πλάκα και, μέσω αυτής, είτε στην επάνω αυλάκωση του οδηγού (όπως και στο σχήμα 4.4.2), είτε, εναλλακτικά, στην κάτω αυλάκωση, αντικαθιστώντας το εξωτερικό πώμα.

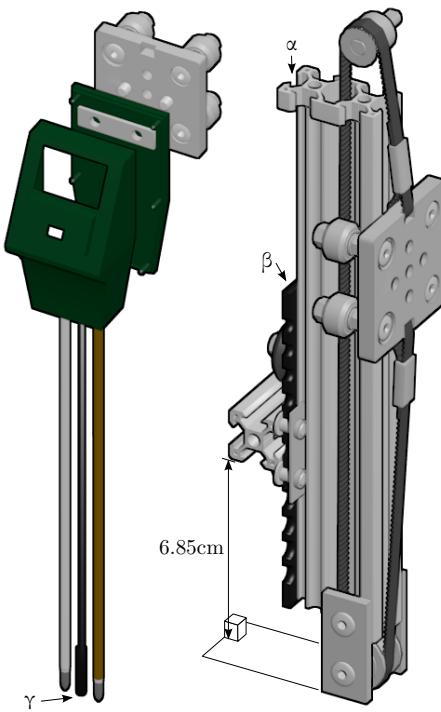
#### 4.4.3 Άξονας Z

Τελικά, στην πλάκα φορτίου του άξονα X στερεώνεται το κατώτερο τμήμα οδηγού για την κίνηση στον άξονα Z, με το μεγαλύτερο τμήμα του να εκτείνεται πάνω από την πλάκα. Και σε αυτήν την περίπτωση επιλέγεται η χρήση μάντας σε συνδυασμό με κινητήρια και ελεύθερη τροχαλία. Επειδή ενδιαφέρει η μετακίνηση μόνο των αισθητήρων που προορίζονται για την παρακολούθηση του υλικού, επιλέγεται η πλάκα Mini V, η οποία χαρακτηρίζεται από πολύ μικρό μέγεθος, και πάνω σε αυτήν προσαρτώνται οι αισθητήρες, όπως φαίνεται αριστερά στο σχήμα 4.4.3. Η εμφανιζόμενη θήκη αποτελεί μέρος των προμηθευμένων αισθητήρων υγρασίας και οξύτητας η οποία έχει τροποποιηθεί ώστε να στεγάζεται ένας ακόμα αισθητήρας, αυτός της θερμοκρασίας (γ), και για τη δημιουργία ορισμένων πρόσθετων οπών για τους κοχλίες και τις γραμμές σήματος.

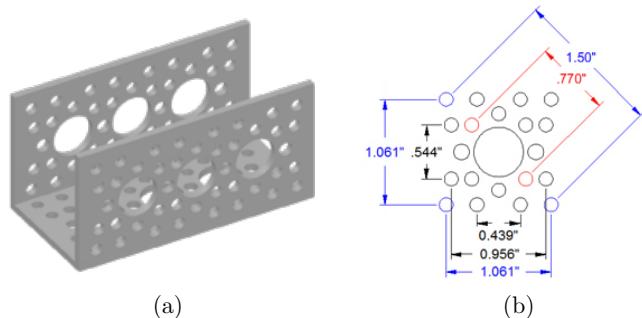
Δεξιά του ίδιου σχήματος εμφανίζεται η διάταξη των στοιχείων που συνθέτουν τον οδηγό Z. Όπως γίνεται αντιληπτό και από το σχήμα, οι βίδες και τα

Σχήμα 4.4.3: Αναπαράσταση οδηγού άξονα Z.

Το μήκος του οδηγού είναι ενδεικτικό και όχι αντιπροσωπευτικό του πραγματικού. Σημειώνεται ότι στο σχήμα δεξιά εμφανίζεται η διατομή ορισμένων στοιχείων.



Σχήμα 4.5.1: Κανάλι Actobotics και διάταξη οπών.



(β'): *Aluminum Channel Schematics* (2014). url: [http://www.actobotics.com/html/3\\_00\\_aluminum\\_channel\\_585442.html](http://www.actobotics.com/html/3_00_aluminum_channel_585442.html) (επίσκεψη 14/04/2014). Actobotics

προσαρτήματα για τη στερέωση του οδηγού που βρίσκονται χαμηλά στην οπίσθια αυλάκωσή του (α), αποτέλουν τη διέλευση του υμάντα από αυτήν. Για το λόγο αυτό επιλέγεται οδηγός VSlot 40 ώστε, εναλλακτικά της αυλάκωσης, να αφιερώνεται η ενδιάμεση σήραγγα ως δίοδος επιστροφής του υμάντα.

Όπως φαίνεται και στο σχήμα, η ελεύθερη τροχαλία στερεώνεται λίγο διαφορετικά σε σχέση με τους τρόπους που έχουν προηγουμένως αναφερθεί, αντικαθιστώντας την ειδική πλάκα με δύο απλά προσαρτήματα. Ο λόγος είναι η αξιοποίηση περισσότερου μήκους των αισθητήρων, εφόσον τα απλά προσαρτήματα δεσμεύουν λιγότερο χώρο της αυλάκωσης στην οποία κινείται και η πλάκα Mini V με αποτέλεσμα, η τελευταία, να μετακινείται χαμηλότερα στον άξονα Z.

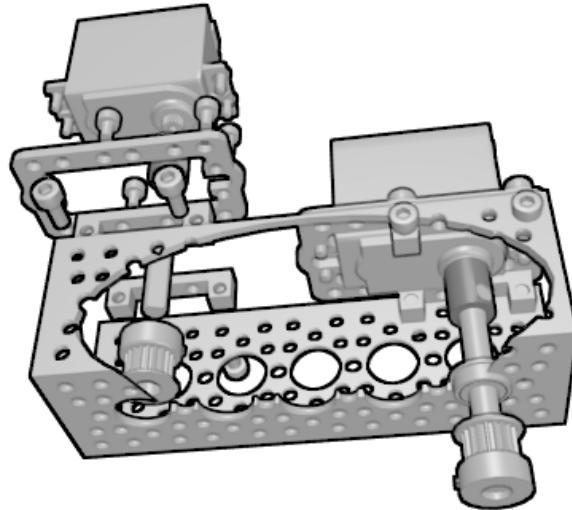
Δεδομένου ότι η πλάκα φορτίου του άξονα X (β) εκτείνεται κάτω από τον οδηγό της, πρέπει να εξασφαλίζεται ότι και αυτή καθώς και τα υπόλοιπα μέρη, κινούνται χωρίς να παρεμποδίζονται από τρίτα αντικείμενα. Αυτό αιτιολογεί τη χρήση αρκετά πλατύ οδηγού για το πλαίσιο της βάσης, όπως αναφέρεται στην ενότητα 4.3.

## 4.5 Τοποθέτηση κινητήρων

Ενώ το σύστημα κατασκευής OpenBuilds παρέχει πολλά εξαρτήματα για την κάλυψη πληθώρας αναγκών, έχει, ωστόσο, αποβεί αδύνατος ο εντοπισμός κάποιων που να διευκολύνουν την τοποθέτηση κινητήρων servo. Αντιμέτως, το σύστημα Actobotics είναι προσανατολισμένο προς τέτοιους κινητήρες. Μολονότι τα δύο συστήματα παρουσιάζουν χαμηλή συμβατότητα στα εξαρτήματά τους, μπορούν να συνυπάρξουν, έως ένα βαθμό.

Η λύση εντοπίζεται στις κυκλικά διατεταγμένες οπές των καναλιών – αντίστοιχων μονάδων των οδηγών VSlot – οι οποίες απέχουν 0.77in, περίπου 2cm,

Σχήμα 4.5.2: Διάταξη κινητήρων αξόνων X και Y.



(σχήμα 4.5.1β), δηλαδή όσο και οι οπές πλακών και περικοχλίων του OpenBuilds, με αποτέλεσμα να μπορούν να συνδεθούν στα σημεία αυτά. Επομένως, επιλέγεται να χρησιμοποιηθούν εξαρτήματα Actobotics για την πλαισίωση των κινητήρων και, ως βάση, το κανάλι του εν λόγω συστήματος το οποίο στη συνέχεια προσδένεται σε κάποιο εξάρτημα του συστήματος OpenBuilds. Στην περίπτωση του κινητήρα Z, το κανάλι προσδένεται απευθείας σε αυλάκωση του αντίστοιχου οδηγού ενώ στην περίπτωση των X και Y, πάνω στην πλάκα γεφυρώματος.

Μία κατηγορία εξαρτημάτων του συστήματος Actobotics ιδιαίτερου ενδιαφέροντος είναι τα στηρίγματα κινητήρα, τα οποία επιτρέπουν πολλές και σύνθετες διατάξεις τους. Μολονότι παρέχεται στήριγμα που στερεώνεται απευθείας στις γωνιακές οπές του καναλιού, επιλέγεται στήριγμα που απαιτεί πρόσθετα γωνιακά προσαρτήματα ώστε ο κινητήρας να εξωθείται ελαφρώς υψηλότερα από το κανάλι προκειμένου να αξιοποιείται κατά το μέγιστο δυνατό ο χώρος στο εσωτερικό του καναλιού. Η ανάγκη αυτή προκύπτει για τον κινητήρα του άξονα X όπου η τροχαλία βρίσκεται στο εσωτερικό του καναλιού. Για περισσότερη ομοιομορφία, επιλέγεται η ίδια διάταξη για τους κινητήρες, ανεξαρτήτως του αν υφίσταται αντίστοιχη επιταχτική ανάγκη ή όχι.

Στο σχήμα 4.5.2 παρουσιάζεται η διάταξη των στοιχείων για τους κινητήρες των αξόνων X και Y. Στην άτρακτο του κινητήρα προσαρτάται σύνδεσμος και, δια μέσω αυτού, το ένα όχρο ράβδου περιστροφής. Για τους κινητήρες X και Z, η ράβδος είναι αρκετά μακριά ώστε να διέρχεται από ένσφαιρο τριβέα τοποθετημένο σε οπή 0.5in του καναλιού. Ο επιπρόσθετος τριβέας αυξάνει την σταθερότητα της

ράβδου, ιδίως όταν στα άκρα της εφαρμόζεται ακτινικό φορτίο (μέσω του ψάντα).

Η διατομή της ράβδου είναι σχήματος D και όχι κυκλική ώστε να δημιουργείται μία επίπεδη επιφάνεια η οποία εξυπηρετεί την ασφαλέστερη πρόσδεση εξαρτημάτων που διαθέτουν ένθετο κοχλία σύσφιξης. Τόσο ο σύνδεσμος όσο και η κινητήρια τροχαλία επωφελούνται από αυτήν την ιδιότητα της ράβδου.



## Κεφάλαιο 5

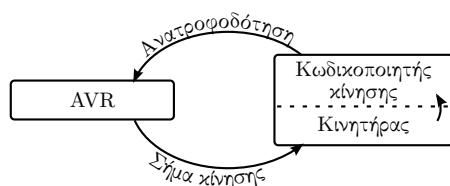
# Κωδικοποιητής κίνησης

Προκειμένου να παρέχεται στο μικροελεγκτή μία ένδειξη για την πορεία της κίνησης του κάθε κινητήρα, χρίνεται σκόπιμη η ύπαρξη μίας μορφής ανατραφοδότησης· ενός αισθητήρα που παρακολουθεί κάποιο φυσικό φαινόμενο που σχετίζεται με την κίνηση του κινητήρα (βλ. σχήμα 5.0.1).

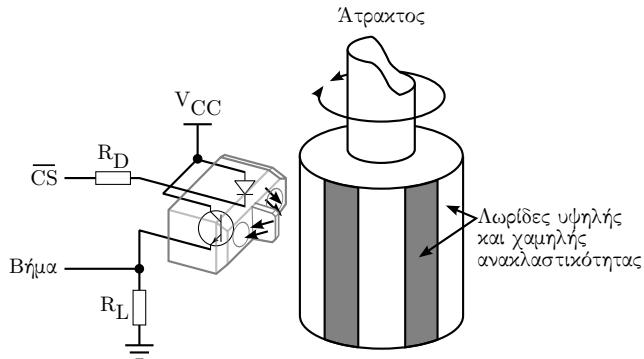
Παρότι υπάρχουν διάφορα φαινόμενα που μπορούν να χρησιμοποιηθούν για αυτόν το σκοπό (όπως μαγνητισμός, ηλεκτρική αντίσταση σε μορφή ποτενσιόμετρου), επιλέγεται η χρήση των, πλέον ίσως όχι και τόσο διαδεδομένων, υπερύθρων ακτίνων. Το επιδιωκόμενο αποτέλεσμα παρουσιάζεται στο σχήμα 5.0.2.

Όπως φαίνεται στο σχήμα, ο αισθητήρας αποτελείται από έναν πομπό και ένα δέκτη υπερύθρων ακτίνων (φωτοδίοδος και φωτοτρανζίστορ, αντίστοιχα). Η άτρακτος είναι ο άξονας περιστροφής του κινητήρα. Πάνω σε αυτόν προσκολλάται μία ειδικά σχεδιασμένη ταινία, η οποία καθώς περιστρέφεται ως αποτέλεσμα περιστροφής της ατράκτου, επηρεάζει την ένταση των εκπεμπόμενων ακτίνων του πομπού που καταφύγανον στο δέκτη, η οποία, με τη σειρά της, επηρεάζει την ένταση της εξόδου του. Με αυτόν τον τρόπο επιτυγχάνεται η μετατροπή την στροφικής κίνησης σε ηλεκτρικούς παλμούς οι οποίοι επιστρέφονται, ιδανικά, ως τετραγωνικοί παλμοί.

Σχήμα 5.0.1: Επίδραση και ανάδραση μικροελεγκτή και κινητήρα.



Σχήμα 5.0.2: Σχηματική απεικόνιση κωδικοποιητή υλοποίησης.



Ο αυτοσχέδιος κωδικοποιητής της υλοποίησης πρόκειται για έναν προσαυξητικό κωδικοποιητή, δηλαδή του οποίου η έξοδος πληροφορεί για την πραγματοποίηση ενός μικρού βήματος από μία λωρίδα στην επόμενη. Σε σχέση με τους απόλυτους κωδικοποιητές που πληροφορούν τη συγκεκριμένη θέση (πιθανώς, σε μοίρες) στην οποία βρίσκεται η άτρακτος, έχει το πλεονέκτημα ότι είναι πολύ πιο απλός στην υλοποίηση και τη διασύνδεση με το μικροελεγκτή, καθώς απαιτεί μόνο έναν αισθητήρα και, συνεπώς, μία γραμμή σύνδεσης για κάθε κινητήρα. Περισσότερα αναφέρονται στην ενότητα Αναγνώριση θέσης (σ. 69).

Ωστόσο, σε αντίθεση με τη συνήθη μορφή, η υλοποίηση χρησιμοποιεί μία ταινία που εφάπτεται της άτρακτου αντί δίσκου. Η επιλογή αυτή γίνεται καθαρά για διευκόλυνση της τοποθέτησης των εξαρτημάτων, παρά την ενδεχόμενη μείωση της ακρίβειας του κωδικοποιητή.

Η επιλογή του υλικού της ταινίας, το πλάτος των λωρίδων της, η απόσταση και η διάταξη (κάθετα ή παράλληλα) του αισθητήρα σε σχέση με αυτές καθώς και η γωνιακή ταχύτητα της άτρακτου επηρεάζουν την ικανότητα αναγνώρισης των μεταβολών από τη μία λωρίδα στην επόμενη (βλ. Συντελεστής σύζευξης σ. 71). Επιπρόσθετα στοιχεία που επηρεάζουν την ακρίβεια του αισθητήρα και, για την ακρίβεια, την πιθανότητα εσφαλμένης εξόδου από το δέκτη (φωτοτρανζίστορ) είναι το αναφερόμενο ως ρεύμα ηρεμίας (dark current) και παρεμβολές από περιβάλλουσες φωτεινές πηγές, καθώς και η θερμοκρασία εν γένει. Αυτές οι, δευτερευούσης σημασίας, παράμετροι αναφέρονται στη σελίδα 78.

Στην ενότητα Υπολογισμοί (σ. 80) γίνεται μία προσπάθεια εκτίμησης των ιδιοτήτων των στοιχείων που απαρτίζουν κάθε κωδικοποιητή ώστε να παραχθεί το επιψυμητό αποτέλεσμα. Μέρος αυτών των υπολογισμών αφιερώνεται στο προσδιορισμό της αντίστασης των αντιστατών  $R_D$  και  $R_L$  του σχήματος, εκ των οποίων ο πρώτος ελέγχει την ένταση που διαρρέει τη φωτοδίοδο, ενώ ο δεύτερος, χρη-

Σχήμα 5.1.1: Προσαυξητικός οπτικός κωδικοποιητής με χρήση φωτοδιακόπτη.



Tycho (2008). *Inkrementalgeber mit Gabellichtschranke*. url:  
[http://commons.wikimedia.org/wiki/File:Inkrementalgeber\\_mit\\_gabellichtschranke.JPG](http://commons.wikimedia.org/wiki/File:Inkrementalgeber_mit_gabellichtschranke.JPG)  
(επίσκεψη 14/04/2014)

σιμεύει για την παραγωγή δύο διαχριτών τιμών (λογικό 0 και 1) ως έξοδο του κωδικοποιητή.

## 5.1 Οπτικοί κωδικοποιητές

Σύμφωνα με έκδοση της DRC (1976, σ. 12), οπτικοί κωδικοποιητές περιστροφικής κίνησης, παραδοσιακά, κατασκευάζονται με την προσάρτηση ενός περιψεριακού διάτρητου δίσκου στον άξονα κίνησης εκατέρωθεν του οποίου διατάσσεται αντικριστό ζεύγος πομπού και δέκτη υπέρυθρων ακτίνων. Καθώς ο δίσκος περιστρέφεται ως αποτέλεσμα κίνησης του άξονα, η ύπαρξη ή έλλειψη οπής επαναφέρει ή αποκόπτει την επικοινωνία μεταξύ πομπού-δέκτη προκαλώντας εναλλαγές στην έξοδο του δέκτη (DRC, 1976, σ. 12).

### 5.1.1 Αναγνώριση θέσης

#### Προσαυξητικοί

Στην απλούστερη υλοποίηση, ο αισθητήρας αναγνωρίζει τη μετάβαση από τη μία θέση στην επόμενη ενώ είναι αδύνατο να αναχθεί από το σήμα και μόνον, είτε η φορά περιστροφής είτε η τρέχουσα γωνιακή μετατόπιση του άξονα. Ο ελεγκτής είναι υπεύθυνος για την εξαγωγή αυτών των συμπερασμάτων (Lynch και Peshkin, 2002,

σσ. 5–6. DRC, 1976, σ. 13). Στην περίπτωση αυτή, ο κωδικοποιητής αποκαλείται προσαυξητικός (incremental) (Lynch και Peshkin, 2002, σ. 5). Ένα παράδειγμα προσαυξητικού κωδικοποιητή περιστροφικής κίνησης παρουσιάζεται στην εικόνα 5.1.1.

### Απόλυτης θέσης

Ωστόσο, είναι δυνατό να κατασκευαστεί απόλυτος (absolute) κωδικοποιητής μετατόπισης, κάνοντας χρήση πολλαπλών ζευγών πομπού-δέκτη και ενός δίσκου υποδιαιρεμένου σε διαχριτές θέσεις που αποτελούνται από μοναδικό συνδυασμό οπών (Lynch και Peshkin, 2002, σ. 6). Ο κάθε αισθητήρας παράγει έξοδο ανεξάρτητη από τους υπολοίπους βάσει των οπών που του αντιστοιχούν, ενώ η συνδυαστική έξοδος όλων των αισθητήρων περιγράφει τον τρέχοντα συνδυασμό οπών και συνεπώς τη γωνιακή μετατόπιση του δίσκου (Lynch και Peshkin, 2002, σ. 6).

### 5.1.2 Διάταξη στοιχείων

Για τη σύνθεση ενός κωδικοποιητή που κάνει χρήση οπτικών αισθητήρων, χρησιμοποιούνται ζεύγη πομπού και δέκτη υπέρυθρων ακτίνων. Το κάθε ζεύγος μπορεί να αποτελείται από ανεξάρτητα, μεταξύ τους, στοιχεία ή να βρίσκονται ενσωματωμένα σε ειδική θήκη που διευκολύνει την τοποθέτησή τους (όπως στην περίπτωση της εικόνας 5.1.1).

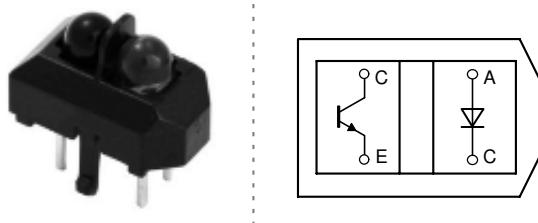
### Φωτοδιακόπτες

Τπάρχουν διατάξεις που τοποθετούν αντικριστά το ζεύγος πομπού και δέκτη σηματίζοντας έναν κενό χώρο μεταξύ τους στον οποίο μπορεί να εισέρχεται εξωτερικό αντικείμενο, διακόπτοντας την επικοινωνία τους. Τέτοιοι αισθητήρες αναφέρονται ως φωτοδιακόπτες (photointerrupter) (Lynch και Peshkin, 2002, σ. 3) και αποτελούν τη διάταξη που έχει παρουσιαστεί μέχρι τώρα.

### Ανακλαστικοί

Σε εναλλακτική διάταξη, πομπός και δέκτης είναι μεταξύ τους παρακείμενοι με την επικοινωνία τους να είναι δυνατή μόνο εφόσον οι εκπεμπόμενες ακτίνες ανακλαστούν σε εξωτερική επιφάνεια (σχήμα ??). Τέτοιοι αισθητήρες αναφέρονται ως ανακλαστικοί (reflective) (Lynch και Peshkin, 2002, σ. 3). Η έξοδος του δέκτη επηρεάζεται όμεσα από την ένταση των προσπίπουσων ακτίνων η οποία, με τη σειρά της, εξαρτάται από τις ανακλαστικές ιδιότητες και την απόσταση της εξωτερικής επιφάνειας (Vishay, 2006).

Σχήμα 5.2.1: Ο ανακλαστικός οπτικός αισθητήρας TCRT5000.



*Reflective Optical Sensor with Transistor Output.* Στο Vishay Semiconductors. *TCRT5000, TCRT5000L*. Έκδ. 1.7. 17 Αύγ. 2009. 12 σσ. url: <http://www.vishay.com/docs/83760/tcr5000.pdf> (επίσκεψη 20/02/2014)

Για την κατασκευή οπτικού κωδικοποιητή κάνοντας χρήση αισθητήρα τέτοιας διάταξης, ο προσαρτημένος στον άξονα περιστροφής δίσκος είναι χωρισμένος σε τμήματα διαφορετικού και εναλλασσόμενου συντελεστή ανάκλασης ώστε με την περιστροφή του να επηρεάζεται η ένταση των προσπίπτουσων ακτίνων στον κάθετο ως προς το δίσκο αισθητήρα, και, συνεπώς, η έξοδός του (Vishay, 2002, σ. 11).

## 5.2 Ανακλαστικός αισθητήρας TCRT5000

Η ενότητα ασχολείται με τις διάφορες παραμέτρους που επηρεάζουν την απόδοση ενός ανακλαστικού αισθητήρα προκειμένου να προσδιοριστούν οι απατήσεις για τη συνδεσμολογία του με το μικροελεγκτή και την τοποθέτησή του σε σχέση με τους κινητήρες με γνώμονα την κάλυψη των αναγκών για την παρακολούθηση της κίνησης των τελευταίων.

Στην περίπτωση του συγκεκριμένου αισθητήρα, TCRT5000, ο δέκτης πρόκειται για ένα φωτοτρανζίστορ. ένα φωτοευαίσθητο ημιαγωγό όπου η ένταση διέλευσης ρεύματος μεταξύ συλλέκτη (collector) και εκπομπού (emitter) ελέγχεται από την ένταση των προσπίπτουσων ακτίνων, ενώ ο πομπός, μία φωτοδίοδος. ένα ημιαγωγό που προκαλεί την παραγωγή φωτός (στην προκειμένη υπέρυθρου) όταν διαρρέεται από ρεύμα. Όπως φαίνεται και στο σχήμα 5.2.1, τα δύο στοιχεία (φωτοδίοδος και φωτοτρανζίστορ) είναι απομονωμένα μεταξύ τους από ένα ενδιάμεσο τοίχωμα. Η επικοινωνία τους καθίσταται δυνατή μόνο εφόσον τα πλησιάσει εξωτερικό αντικείμενο.

### 5.2.1 Συντελεστής σύζευξης

Σύμφωνα με τη Vishay (2002), στους ανακλαστικούς αισθητήρες που κάνουν χρήση φωτοτρανζίστορ, ο λόγος της έντασης ρεύματος του συλλέκτη προς το ρεύμα ορθής φοράς,  $\frac{I_C}{I_F}$ , αναφέρεται ως συντελεστής σύζευξης (coupling factor),

**Πίνακας 5.2.1: Σχετική απόδοση διάφορων ανακλαστικών υλικών.**

Σε όλες τις μετρήσεις, το ρεύμα ορθής φοράς,  $I_F$ , ήταν σταθερό στα 20mA, ο αισθητήρας τοποθετημένος κάνθετα ως προς την ανακλαστική επιφάνεια σε απόσταση όπου ο συλλέκτης αποδίδει τη μέγιστη έξοδο για υπέρυθρες των 950nm Vishay (2006).

<b>Kodak neutral card</b>		<b>Black on white typewriting paper</b>	
White side (reference medium)	100%	Drawing ink (Higgins, Pelikan)	4–6%
Gray side	20%	Foil ink (Rotring)	50%
<b>Paper</b>		<b>Plotter pen</b>	
Typewriting paper	94%	Fiber-tip pen (Edding 400)	10%
Drawing card, white (Schoeller)	100%	Fiber-tip pen, black (Stabilo)	76%
Card, light gray	67%	Photocopy	7%
Envelope (beige)	100%	HP fiber-tip pen (0.3mm)	84%
Packing card (light brown)	84%	Black 24 needle printer	28%
Newspaper paper	97%	Ink (Pelikan)	100%
Pergament paper	30–42%	Pencil, HB	26%

Table 1. Στο Vishay Semiconductors. *Application of Optical Sensors*. 27 Σεπτ. 2006. 7 σσ. url: <http://www.vishay.com/docs/80107/80107.pdf> (επίσκεψη 01/03/2014), σ. 2

$k$ , και περιγράφει το βαθμό οπτικής σύνδεσης μεταξύ πομπού και δέκτη. Ο προσδιορισμός του γίνεται για ορισμένη ανακλαστική επιφάνεια και απόσταση από αυτήν και επηρεάζεται από την ένταση ρεύματος του πομπού, τη θερμοκρασία και τη συχνότητα εναλλαγής μεταξύ επιφανειών διαφορετικών συντελεστών ανάκλασης (Vishay, 2002).

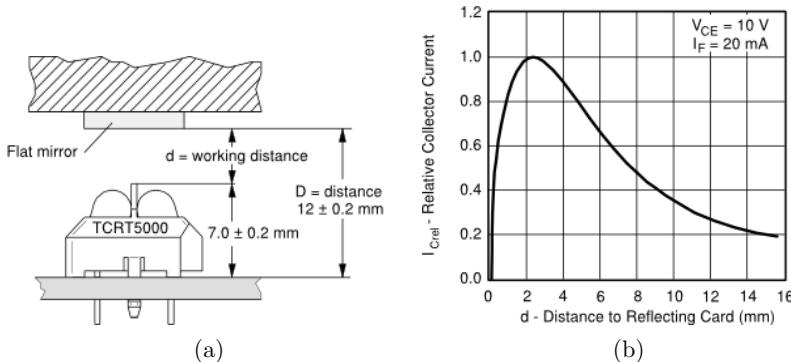
### Ανακλαστική επιφάνεια

Ο πίνακας 5.2.1, ο οποίος αποτελεί απόσπασμα μετρήσεων της Vishay (2006), παρουσιάζει το ποσοστό της έντασης ρεύματος που σημειώνεται στο συλλέκτη για διάφορα ανακλαστικά υλικά σε σχέση με τη χρήση της λευκής όψης κάρτας Kodak neutral (No.Q-13). Για τις ανάγκες της υλοποίησης, η επιφάνεια κωδικοποίησης είναι αρκετό να αποτελείται από διαδοχικά τμήματα που παρουσιάζουν μεγάλη απόκλιση στους συντελεστές ανάκλασης, ώστε να μην παρατηρείται επικάλυψη στο εύρος έντασης ρεύματος του συλλέκτη για καθένα και, συνεπώς, να αυξάνεται η δυνατότητα διώχρισή τους. Μία δεύτερη απαίτηση είναι η διαθεσιμότητα και η ευχρηστία των αντίστοιχων υλικών ώστε να είναι άμεση η ενδεχόμενη αντικατάσταση ή τροποποίηση της επιφάνειας κωδικοποίησης.

Με αυτά τα κριτήρια, επιλέγεται το απλό τυπογραφικό χαρτί ως επιφάνεια υψηλού συντελεστή ανάκλασης (94%) με την επικάλυψη του με φωτοτυπικό μελάνι ως επιφάνεια χαμηλού συντελεστή (7%).

Για περαιτέρω απλούστευση της υλοποίησης, επιλέγεται η αντικατάσταση του δίσκου κωδικοποίησης με ταινία η οποία καλύπτει την περιφέρεια του άξονα περι-

Σχήμα 5.2.2: Λειτουργική απόσταση και λειτουργικό διάγραμμα.



(α'): Fig. 3 - Test Circuit. Στο Vishay Semiconductors. TCRT5000, TCRT5000L. Εκδ. 1.7. 17 Αύγ. 2009. 12 σσ. url: <http://www.vishay.com/docs/83760/tcrt5000.pdf> (επίσκεψη 20/02/2014), σ. 3

(β'): Fig. 9 - Relative Collector Current vs. Distance. Στο Vishay Semiconductors. TCRT5000, TCRT5000L. Εκδ. 1.7. 17 Αύγ. 2009. 12 σσ. url: <http://www.vishay.com/docs/83760/tcrt5000.pdf> (επίσκεψη 20/02/2014), σ. 4

στροφής στο σημείο όπου είναι τοποθετημένος ο αισθητήρας.

### Λειτουργική απόσταση

Η απόσταση του αισθητήρα από την ανακλαστική επιφάνεια επηρεάζει άμεσα το ποσοστό των προσπίπτουσων ακτίνων στο δέκτη που είναι υπεύθυνες για τη διέγερση του φωτοτρανζίστορ. Η απόσταση αυτή αποκαλείται λειτουργική απόσταση (operating distance),  $d$ , και, για μία τυπική υλοποίηση, απεικονίζεται στο σχήμα 5.2.2(α') (Vishay, 2002).

Όπως είναι αναμενόμενο, καθώς η λειτουργική απόσταση μεταβάλλεται, η ένταση ρεύματος του συλλέκτη αυξομειώνεται. Σύμφωνα με τη τον οδηγό της Vishay (2006), η σχέση αυτή απεικονίζεται στο λειτουργικό διάγραμμα (operating diagram) στο εγχειρίδιο χρήσης κάθε αισθητήρα.

Το σχήμα 5.2.2(β') αποτελεί το λειτουργικό διάγραμμα του επιλεγμένου αισθητήρα, TCRT5000, όπου παρουσιάζεται η ένταση ρεύματος του συλλέκτη,  $I_C$ , σε σχέση με τη μέγιστη δυνατή,  $I_{Cmax}$ , καθώς μεταβάλλεται η λειτουργική απόσταση. Επίσης, παρατηρείται ότι η μέγιστη ένταση του συλλέκτη,  $I_C = I_{Cmax}$ , σημειώνεται για μία λειτουργική απόσταση  $d = 2.5$  cm.

Για τις ανάγκες της υλοποίησης, επιλέγεται λειτουργική απόσταση 2.5 cm έτσι ώστε ο συντελεστής σύζευξης να ευνοείται όσο το δυνατόν περισσότερο.

Σχήμα 5.2.3: Σχέση μεταβολής ρεύματος  $I_C$  και συντελεστή ανάλλασης.

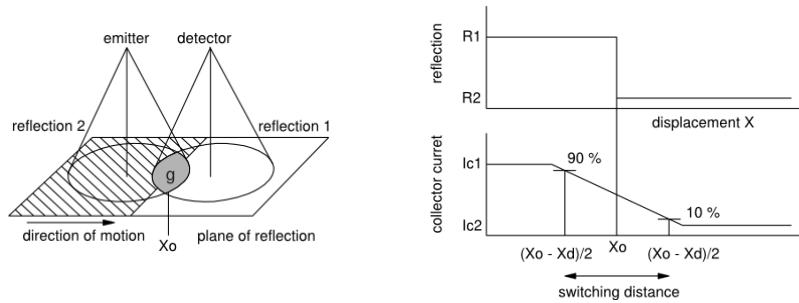


Figure 4. Abrupt reflection change with associated  $I_C$  curve. Στο Vishay Semiconductors. Application of Optical Sensors. 27 Σεπτ. 2006. 7 σσ. url: <http://www.vishay.com/docs/80107/80107.pdf> (επίσκεψη 01/03/2014), σ. 3

### Διάστημα εναλλαγής

Για τις ανάγκες της υλοποίησης, ο άξονας περιστροφής επικαλύπτεται, στο ύψος του αισθητήρα, από διαδοχικά τμήματα υψηλότερου και χαμηλότερου συντελεστή ανάλλασης. Καθώς ο άξονας περιστρέφεται, η ένταση ρεύματος του συλλέκτη μεταβάλλεται από τη μέγιστη μέχρι την ελάχιστη δυνατή και αντίστροφα. Ωστόσο, σύμφωνα με το εγχειρίδιο της Vishay (2006), εάν το πάχος των τμημάτων είναι πολύ μικρό, ενδέχεται οι εναλλαγές να μην εκδηλώνονται αισθητά στην ένταση ρεύματος του συλλέκτη.

Στο σχήμα 5.2.3 παρουσιάζεται η ένταση ρεύματος του συλλέκτη για δύο διαδοχικά τμήματα. Το σημείο εναλλαγής από τον ένα συντελεστή ανάλλασης στον επόμενο σημειώνεται με το  $X_0$  ενώ με  $I_{c1}$  και  $I_{c2}$ , η μέγιστη ένταση ρεύματος του συλλέκτη όταν η κοινή επιφάνεια  $g$  των οπτικών πεδίων πομπού και δέκτη καλύπτεται πλήρως από τμήμα του αντίστοιχου συντελεστή. Προκύπτει ότι, ενώ η μετάβαση από τον ένα συντελεστή στον επόμενο συμβαίνει ακαριαία, η ένταση ρεύματος του συλλέκτη  $I_C$  έχει αρχίσει να μειώνεται σε προγενέστερη μετατόπιση, όταν ένα πρώτο τμήμα των αρχικά διαθέσιμων ακτίνων αποκόπηκε από το δέκτη, και συνεχίζει να μειώνεται σταδιακά έως ότου το τμήμα με το νέο συντελεστή έχει καλύψει πλήρως την περιοχή  $g$ .

Επίσης, συμπεραίνεται ότι, καθώς το πάχος των τμημάτων μειώνεται ώστε η περιοχή  $g$  να είναι αδύνατο να καλυφθεί εξ ολοκλήρου από ένα μόνο τμήμα, η μέγιστη και ελάχιστη τιμή έντασης ρεύματος που είναι δυνατό να σημειωθούν αρχίζουν να συγχλίνουν, εφόσον, πλέον, σε κάθε μετατόπιση καταφθάνουν στο δέκτη ακτίνες από όλο και περισσότερα τμήματα.

Το ελάχιστο πάχος τμημάτων καθορίζεται από το διάστημα εναλλαγής (switching distance),  $X_d$ , το οποίο ορίζεται ως το διάστημα μεταξύ δύο διαδοχικών τμημάτων

Σχήμα 5.2.4: Σχέση Λειτουργικής απόστασης και Διαστήματος εναλλαγής.

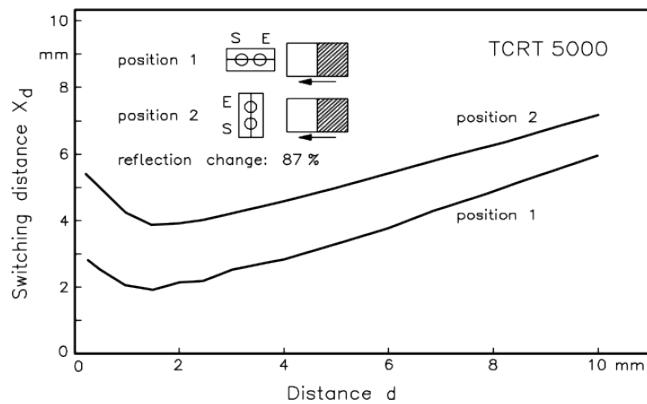


Figure 15. The switching distance as a function of the distance  $A$  for the reflex sensors TCRT5000, CNY70 and TCRT1000. Στο Vishay Semiconductors. Application of Optical Reflex Sensors. TCRT1000, TCRT5000, CNY70. Φεβ. 2002. 17 σσ. url: <http://www.vishay.com/docs/81449/81449.pdf> (επίσκεψη 10/03/2014), σ. 8

διαφορετικών συντελεστών ανάκλασης στο οποίο παρατηρείται το 90%  $I_{C_1}$  έως το 10%  $I_{C_2}$  (Vishay, 2006). Η τιμή του διαστήματος εναλλαγής εξαρτάται, από την κατασκευή του αισθητήρα καθώς και τη λειτουργική απόσταση, ενώ όσο πιο μικρή απόσταση εναλλαγής υποστηρίζει κάποιοις αισθητήρες, τόσο υψηλότερη η διακριτική ικανότητα (resolution) του (Vishay, 2002. Vishay, 2006). Για τον αισθητήρα TCRT5000, το διάστημα εναλλαγής ανέρχεται στα 1.9mm (Vishay, 2002).

Στο σχήμα 5.2.4 παρουσιάζεται πώς η λειτουργική απόσταση του αισθητήρα επηρεάζει το διάστημα εναλλαγής του. Οι εμφανιζόμενες καμπύλες αποτελούν ένα κατώτατο όριο και αντιστοιχούν σε δύο διαφορετικούς τρόπους τοποθέτησης του αισθητήρα σε σχέση με την ανακλαστική επιφάνεια. Στη θέση 1, ο νοητός άξονας που ορίζεται από πομπό και δέκτη του αισθητήρα είναι κάθετος ως προς τα εναλλασσόμενα τμήματα, ενώ στη θέση 2, παράλληλος. Παρατηρείται ότι η θέση 1 υπερτερεί της θέσης 2 καθώς για την ίδια λειτουργική απόσταση επιτυγχάνεται υψηλότερη διακριτική ικανότητα και, συνεπώς, προσφέρεται υψηλότερο ελάχιστο διάστημα εναλλαγής. Δεδομένου ότι οι συνθήκες (ανάγκες και περιορισμοί) το επιτρέπουν, επιλέγεται η θέση 1 για την υλοποίηση.

### Συχνότητα αποκοπής

Το φωτοτρανζίστορ—το πιο αργό στοιχείο του αισθητήρα—απαιτεί χάπιο χρόνο για να ανταποκριθεί στις απότομες εναλλαγές έντασης των προσπίπτουσων ακτίνων (Vishay, 2006). Ως αποτέλεσμα, εάν οι νέες εναλλαγές προκύπτουν προτού η έξοδος του να έχει κατασταλάξει για μία προηγούμενη εναλλαγή, τότε η αναγνώρισή τους καθίσταται μάλλον αβέβαιη. Άμεση απόρροια αυτού, δηλαδή

Σχήμα 5.2.5: Σχέση Συχνότητας αποκοπής και Αντίστασης φόρτου.

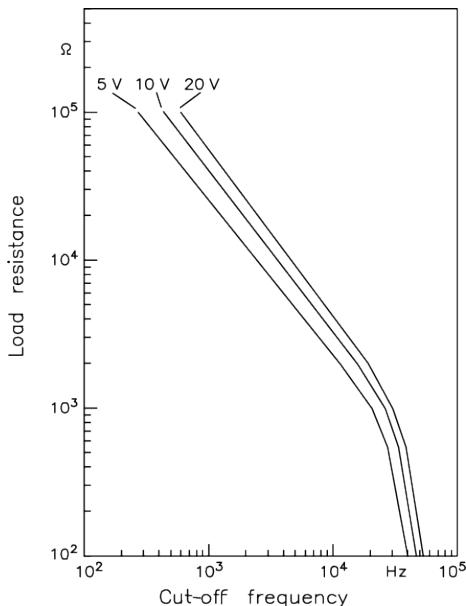


Figure 11. Cut-off frequency,  $f_c$ . Στο Vishay Semiconductors. Application of Optical Reflex Sensors. TCRT1000, TCRT5000, CNY70. Φεβ. 2002. 17 σσ. url: <http://www.vishay.com/docs/81449/81449.pdf> (επίσκεψη 10/03/2014), σ. 5

του χρόνου απόκρισης του φωτοτρανζίστορ, είναι η ανάγκη προσδιορισμού μίας μέγιστης επιτρεπόμενης συχνότητας εναλλαγής των τμημάτων διαφορετικού συντελεστή ανάκλασης ώστε να μην επηρεάζεται δραματικά ο συντελεστής σύζευξης. Η συχνότητα αυτή αναφέρεται ως συχνότητα αποκοπής (cut-off frequency) και είναι η συχνότητα στην οποία παρατηρείται μείωση περίπου 30% του συντελεστή σύζευξης (Vishay, 2002).

Το σχήμα 5.2.5 παρουσιάζει πώς επηρεάζουν η τάση και η αντίσταση φόρτου,  $R_L$ , του φωτοτρανζίστορ τη συχνότητα αποκοπής του αισθητήρα. Θα μπορούσε να χρησιμοποιηθεί χαμηλή τιμή αντίστασης φόρτου, ώστε να μειωθεί ο χρόνος απόκρισης και, ως επέκταση, να διευρυνθεί το όριο της συχνότητας αποκοπής. Ωστόσο, μία τέτοια κίνηση θα είχε ως αποτέλεσμα τη μείωση της τάσης του παραγόμενου σήματος (Vishay, 2006). Προτιμάται να χρησιμοποιηθούν οι απαιτήσεις της υλοποίησης, όπως μέγιστη ταχύτητα περιστροφής, για τον προσδιορισμό της συχνότητας αποκοπής και, μέσω αυτής, της αντίστασης φόρτου.

### Θερμοκρασία

Η αύξηση θερμοκρασίας επηρεάζει την απόδοση τόσο της διόδου εκπομπής υπερύθρων, η οποία μειώνεται, όσο και του φωτοτρανζίστορ, η οποία αυξάνεται

Σχήμα 5.2.6: Σχέση θερμοκρασίας και Σχετικού λόγου μεταφοράς ρεύματος.

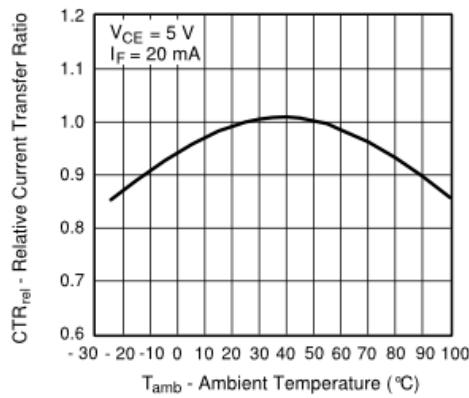


Fig. 3 - Test Circuit. Στο Vishay Semiconductors. TCRT5000, TCRT5000L. Έκδ. 1.7. 17 Αύγ. 2009. 12 σσ. url: <http://www.vishay.com/docs/83760/tcrt5000.pdf> (επίσκεψη 20/02/2014), σ. 3

(Vishay, 2006). Ωστόσο, όπως προκύπτει από το σχήμα 5.2.6, για θερμοκρασίες από 0°C έως 90°C, το συνολικό αποτέλεσμα επηρεάζεται ελάχιστα. Θα μπορούσε, είτε να αγνοηθεί πλήρως είτε να συμπεριληφθεί ως σταθερή μείωση του σχετικού λόγου μεταφοράς ρεύματος.

### Ένταση πομπού

Η ένταση των εκπεμπόμενων ακτίνων εξαρτάται από κάποια χαρακτηριστικά κατασκευής του αισθητήρα (όπως φακός εστίασης) και την ένταση ρεύματος της διόδου υπερύθρων. Αφενός, η δίοδος πρέπει να διαρρέεται από ρεύμα ορθής φοράς ελάχιστης έντασης 5mA ώστε να σταθεροποιείται η έξοδός της, αφετέρου, να μην ξεπερνά τη μέγιστη αποδεκτή τιμή (Vishay, 2002). Σύμφωνα με τις απόλυτες μέγιστες τιμές του αισθητήρα TCRT5000, η μέγιστη ένταση ρεύματος ορθής φοράς,  $I_F$ , είναι 60mA, σε θερμοκρασία περιβάλλοντος χώρου,  $T_{amb}$ , 25°C (Vishay, 2009).

Παρόλο που η εφαρμογή της μέγιστης δυνατής έντασης παρέχει ισχυρότερο σήμα στο δέκτη λόγω του υψηλότερου συντελεστή σύζευξης (σχήμα 5.2.7) προτιμάται η εφαρμογή κατά πολύ χαμηλότερης. Στους λόγους συγκαταλέγεται η τήρηση αποδεκτής έντασης ρεύματος σε περιβάλλον κυμαινόμενης θερμοκρασίας κυρίως για την επιμήκυνση της διάρκειας ζωής του πομπού. Επίσης, στο ίδιο σχήμα παρατηρείται ότι ο συντελεστής σύζευξης πλησιάζει ένα ανώτατο όριο 6% για ρεύμα ορθής φοράς στο εύρος 35–60mA. Επομένως, επιλέγεται να χρησιμοποιηθεί ρεύμα ορθής φοράς έως και 35mA.

Σχήμα 5.2.7: Σχέση Ρεύματος ορθής φοράς και Λόγου μεταφοράς ρεύματος.

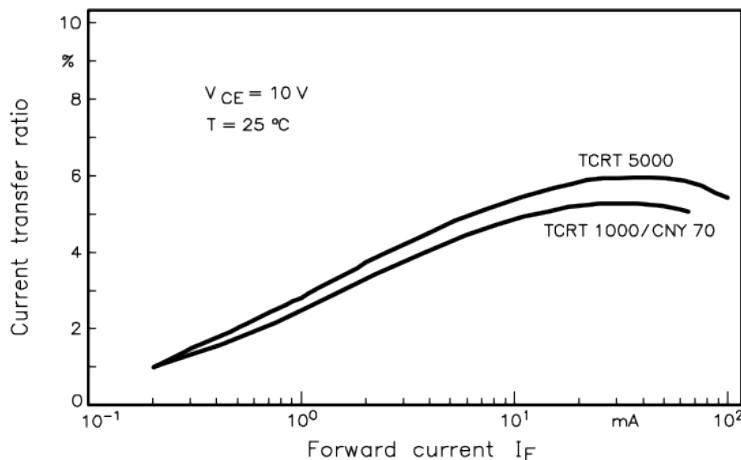


Figure 9. Coupling factor  $k = f(I_F)$  of the reflex sensors. Στο Vishay Semiconductors. Application of Optical Reflex Sensors. TCRT1000, TCRT5000, CNY70. Φεβ. 2002. 17 σσ. url: <http://www.vishay.com/docs/81449/81449.pdf> (επίσκεψη 10/03/2014), σ. 5

### 5.2.2 Λοιπές παράμετροι

#### Ρεύμα ηρεμίας

Ένα φωτοτρανζίστορ διαρρέεται από ρεύμα ακόμα και όταν αυτό βρίσκεται πλήρως απομονωμένο από φωτεινές πηγές. Το ρεύμα αυτό αναφέρεται ως ρεύμα ηρεμίας (dark current) και επηρεάζεται από την τιμή της τάσης συλλέκτη-εκπομπού,  $V_{CE}$ , και, σε μεγαλύτερο βαθμό, από τη θερμοκρασία (Vishay, 2006). Τυπική τιμή ρεύματος ηρεμίας,  $I_{CEO}$ , για τον αισθητήρα TCRT5000 είναι τα 10nA στα 20V (Vishay, 2009).

Κρίνεται σκόπιμο να αγνοηθεί στο σχεδιασμό εφόσον προκύψει από τους υπολογισμούς ότι το ρεύμα ηρεμίας έντασης περίπου 5mA που προκαλείται από την εφαρμογή τάσης 10V σε μία οριακή θερμοκρασία,  $T_{amb}$ , 100°C, επηρεάζει ελάχιστα την έξοδο του αισθητήρα (σχήμα 5.2.8).

#### Οπτικές παρεμβολές

Σύμφωνα με τον οδηγό Vishay (2006), είναι δυνατό να διοχετεύονται εκπεμπόμενες ακτίνες από τον πομπό στο δέκτη απευθείας μέσα από την ίδια τη θήκη καθώς και δια μέσω επιφανειών που περιβάλλουν τον αισθητήρα, πέραν της ανακλαστικής επιφάνειας, με αποτέλεσμα να προκύπτει ένταση ρεύματος συλλέκτη.

Επιπλέον, σταθερή απευθείας πρόσπτωση φωτός στο φωτοτρανζίστορ μειώνει την ευαισθησία του, δυνατό φως είναι δυνατό να το κρατήσει μονίμως ενεργό, ενώ μεταβαλλόμενο, να προκαλέσει αδικαιολόγητη εναλλαγή στο σήμα εξόδου (Vishay,

Σχήμα 5.2.8: Σχέση θερμοκρασίας και Ρεύματος ηρεμίας.

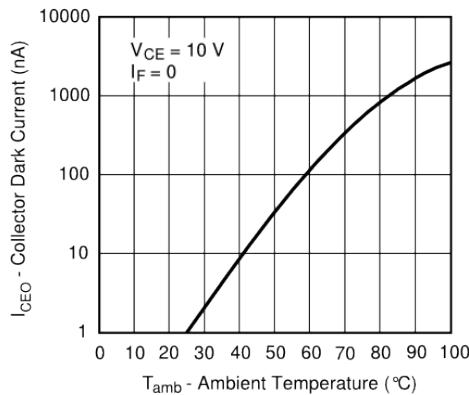


Figure 5. Collector Dark Current vs. Ambient Temperature. Στο Vishay Semiconductors. Application of Optical Sensors. 27 Σεπτ. 2006. 7 σσ. url: <http://www.vishay.com/docs/80107/80107.pdf> (επίσκεψη 01/03/2014), σ. 3

2006). Ο επιλεγμένος αισθητήρας διαθέτει προστατευτικά φίλτρα τα οποία παρεμποδίζουν το ορατό φως (Vishay, 2009). Ωστόσο, ένα μεγάλο εύρος του ηλιακού φωτός αποτελείται από υπέρυθρες και, συνεπώς, η παράμετρος πρέπει να ληφθεί υπόψη κατά το σχεδιασμό του συστήματος.

Η περίπτωση παρεμβολών που οφείλονται στην κατασκευή του αισθητήρα είναι δυνατό να απαλειφθούν καταμετρώντας το σήμα του αισθητήρα σε πλήρη απομόνωσή του και λαμβάνοντάς το υπόψη, ως σταθερά, κατά την κανονική λειτουργία του κωδικοποιητή. Ο βαθμός στον οποίο επηρεάζουν οι περιβάλλοντες επιφάνειες μπορεί να εντοπιστεί με παρόμοιο τρόπο. Ο βαθμός στον οποίο επηρεάζουν οι επικρατούσες συνθήκες φωτισμού είναι, σαφώς, μεταβλητός και η ίδια προσέγγιση μη εφαρμοστέα. Ωστόσο, είναι δυνατό να περιοριστεί απομονώνοντας εξωτερικά ολόκληρο τον κωδικοποιητή, δηλαδή αισθητήρα και ανακλαστική επιφάνεια.

### Θερμοκρασία

Σε προηγούμενη παράγραφο, μελετήθηκε πώς επηρεάζει η θερμοκρασία το συντελεστή σύζευξης. Ωστόσο, η θερμοκρασία θέτει και ορισμένα άλλα ζητήματα προς μελέτη.

Η αύξηση θερμοκρασίας της διόδου, ως άμεσο υποπροϊόν της κατανάλωσης ισχύος, παίζει καθοριστικό ρόλο στον προσδιορισμό της έντασης ρεύματος της διόδου. Σύμφωνα με το εγχειρίδιο χρήσης τους αισθητήρα (Vishay, 2009), η μέγιστη επιτρεπτή τιμή ρεύματος ορθής φοράς του πομπού,  $I_F$ , ανέρχεται στα 60mA σε θερμοκρασία περιβάλλοντος χώρου,  $T_{amb}$ , 25°C. Επιπλέον, καθώς η θερμοκρασία αυξάνεται, το μέγιστο επιτρεπτό όριο μειώνεται. Το σχήμα 5.2.9 δίνει την

Σχήμα 5.2.9: Σχέση Κατανάλωσης ισχύος και Θερμοκρασίας.

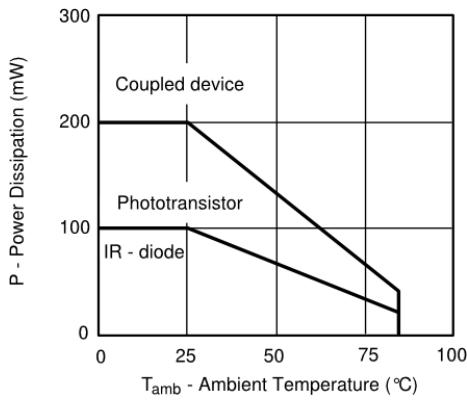


Fig. 1 - Power Dissipation Limit vs. Ambient Temperature. Στο Vishay Semiconductors.

TCRT5000, TCRT5000L. Έκδ. 1.7. 17 Αύγ. 2009. 12 σσ. url:

<http://www.vishay.com/docs/83760/tcr5000.pdf> (επίσκεψη 20/02/2014), σ. 3

απόλυτη μέγιστη τιμή κατανάλωσης ισχύος σε σχέση με τη θερμοκρασία. Λαμβάνοντας υπόψη ότι  $P = VI$ , είναι δυνατό να υπολογιστεί το μέγιστο επιτρεπτό ρεύμα ορθής φοράς βάσει των διακυμάνσεων της θερμοκρασίας χώρου και της εφαρμοζόμενης τάσης της υλοποίησης.

Κρίνεται αναγκαία η τήρηση χαμηλών θερμοκρασιών στη δίοδο για την επιμήκυνση της ζωής της και, συνεπώς, του ίδιου του αισθητήρα, καθώς χαμηλή θερμοκρασία συνεπάγεται ελάχιστη ή καθόλου φιλορά της ένωσης p-n της δίοδου. Προς επίτευξη αυτού, η παρεχόμενη ένταση ρεύματος της διόδου είναι κατά πολύ μικρότερη από τη μέγιστη αποδεκτή, κατάλληλη ακόμα και για θερμοκρασίες που ξεπερνούν τις απαιτήσεις της υλοποίησης. Επίσης, ο αισθητήρας τίθεται σε λειτουργία μόνο για τα διαστήματα περιστροφής του άξονα κίνησης, ώστε ο χρόνος κατανάλωσης ενέργειας με αποτέλεσμα την εκπομπή θερμότητας και αύξηση της θερμοκρασίας τοπικά του αισθητήρα να είναι περιορισμένος.

### 5.2.3 Υπολογισμοί

Σε αυτό το σημείο, επιχειρείται να προσδιοριστούν οι επακριβείς τιμές των στοιχείων για τη σύνδεση του αισθητήρα υπέρυθρων και των τελικών προδιαγραφών του κωδικοποιητή, λαμβάνοντας υπόψη τις παραμέτρους που έχουν ήδη περιγραφεί.

Μία βασική ανάγκη είναι η εκτίμηση της έντασης ρεύματος του συλλέκτη,  $I_C$ , η οποία εξαρτάται από το ρεύμα ορθής φοράς,  $I_F$ , και το συντελεστή σύζευξης,  $k$ .

$$I_C = k \cdot I_F \quad (5.2.1)$$

Ο συντελεστής σύζευξης εξαρτάται από το ρεύμα ορθής φοράς και η σχέση τους περιγράφεται από το σχήμα 5.2.7. Ωστόσο, το σχήμα αυτό αναφέρεται στη χρήση της λευκής όψης κάρτας Kodak neutral σε λειτουργική απόσταση μέγιστης απόδοσης.

Στην υλοποίηση χρησιμοποιούνται διαφορετικά υλικά ως ανακλαστικές επιφάνειες και, ενδεχομένως, διαφορετική λειτουργική απόσταση. Ο πραγματικός συντελεστής σύζευξης για κάθε υλικό και λειτουργική απόσταση προκύπτει από την

$$k = I_{Crel} \cdot k_{Kodak} \quad (5.2.2)$$

όπου  $I_{Crel}$ , η σχετική τιμή ρεύματος του συλλέκτη ως αποτέλεσμα του ανακλαστικού υλικού, της λειτουργικής απόστασης και λοιπών παραμέτρων και  $k_{Kodak}$ , ο συντελεστής σύζευξης με χρήση της κάρτας Kodak neutral.

Το ρεύμα ορθής φοράς,  $I_F$ , είναι επιλυμητό να επιλεγεί ώστε να μεγιστοποιείται ο συντελεστής σύζευξης,  $k$ , και προκύπτει ότι για τον επιλεγμένο αισθητήρα, TCRT5000, πρατηρείται για ρεύμα ορθής φοράς περίπου 35mA (σχήμα 5.2.7). Ωστόσο, λαμβάνοντας υπόψη το σχήμα 5.2.9 και την ανάγκη για τίρηση χαμηλής κατανάλωσης ισχύος με εφαρμογή ορθής τάσης  $V_F = 1.25V$  στη δίοδο IR, αντί αυτού, επιλέγεται ρεύμα ορθής φοράς

$$I_F = 20mA \quad (5.2.3)$$

με

$$k_{Kodak} = 5.5\% \quad (5.2.4)$$

Έχει σημειωθεί ότι, εκτός και εάν προκύψει κάποια ιδιαίτερη ανάγκη, η λειτουργική απόσταση,  $d$ , του αισθητήρα επιλέγεται ώστε ο συντελεστής σύζευξης να ευνοείται κατά το μέγιστο. Επομένως, από το σχήμα 5.2.2, προκύπτει ότι

$$d = 2.5mm \quad (5.2.5)$$

Από το σχήμα 5.2.4 προκύπτει ότι για την επιλεγμένη λειτουργική απόσταση (2.5mm), το ελάχιστο επιτρεπτό διάστημα εναλλαγής είναι περίπου 2mm. Αφήνοντας περιθώριο σφάλματος, επιλέγεται διάστημα εναλλαγής

$$X_d = 4mm \quad (5.2.6)$$

Στην υλοποίηση χρησιμοποιείται τυπογραφικό χαρτί ως επιφάνεια με υψηλό συντελεστή σύζευξης,  $k_H$ , και φωτοτυπικό μελάνι ως επιφάνεια με χαμηλό συντελεστή,  $k_L$ . Λαμβάνοντας υπόψη τον πίνακα ανακλαστικών υλικών (πίνακας 5.2.1),

το λειτουργικό διάγραμμα για απόσταση  $d = 2.5\text{mm}$  (σχήμα 5.2.2β) και τη σχέση (5.2.4), αντικαθιστώντας στη σχέση (5.2.2), προκύπτει ότι

$$k_H = 94\% \cdot 5.5\% = 5.17\% \quad (5.2.7)$$

$$k_L = 7\% \cdot 5.5\% = 0.385\% \quad (5.2.8)$$

Με γνωστούς τους συντελεστές σύζευξης  $k_H$  και  $k_L$  των δύο επιφανειών και το ρεύμα ορθής φοράς (σχέση (5.2.3)), είναι πλέον δυνατός ο υπολογισμός των αντίστοιχων τιμών της έντασης ρεύματος του συλλέκτη, αντικαθιστώντας στη σχέση (5.2.1)

$$I_{C_H} = 5.17\% \cdot 20\text{mA} = 1.034\text{mA} \quad (5.2.9)$$

$$I_{C_L} = 0.385\% \cdot 20\text{mA} = 0.077\text{mA} \quad (5.2.10)$$

Αφήνοντας ένα περιθώριο 10% για μειωμένη απόδοση της διόδου IR ως αποτέλεσμα φυλοράς από την πάροδο χρόνου και ένα 10% λόγω μεταβολών στη θερμοκρασία στο εύρος 0–90°C (σχήμα 5.2.6), η ένταση ρεύματος του συλλέκτη ως αποτέλεσμα των υπερύθρων αναπροσαρμόζεται σε  $I_{C_H} = 0.827\text{mA}$ .

Για το εύρος θερμοκρασιών που ενδιαφέρει, το ρεύμα ηρεμίας είναι περίπου 1μΑ (0–90°C, σχήμα 5.2.8) και αμελητέο για της ανάγκες της υλοποίησης.

Όπως προκύπτει από τους υπολογισμούς, η ένταση ρεύματος του συλλέκτη κυμαίνεται εντός μερικών εκατοντάδων μΑ, ενώ, κύριο ενδιαφέρον είναι το πλήθος των μεταβολών και όχι η ανίχνευση της πραγματικής τιμής της έντασης. Σύμφωνα με δελτίο των OPTEK (2004) και Faichild (2002), το φωτοτρανζίστορ είναι δυνατό να χρησιμοποιηθεί ως διακόπτης (switch mode) όπου το παραγόμενο σήμα του ενισχύεται και ανάγεται σε δυαδικό ψηφίο προσαρμόζοντας μόνο την αντίσταση φόρτου, χωρίς να απαιτείται χρήση επιπρόσθετων ηλεκτρονικών διατάξεων, ως ακολούθως

$$R_L > \frac{V_{CC} - V_{CEsat}}{I_C} \quad (5.2.11)$$

όπου  $V_{CC}$  η τάση της πηγής,  $V_{CEsat}$  η τάση κορεσμού συλλέκτη-εκπομπού και  $I_C$  η ένταση ρεύματος του συλλέκτη. Με αυτόν τον τρόπο, το φωτοτρανζίστορ παραμένει αποκομμένο παράγοντας τάση περίπου 0.8V (λογικό 0), έως ότου κορεστεί από τις προσπίπτουσες ακτίνες ώστε να παρέχει περίπου την τάση της πηγής (λογικό 1) (Faichild, 2002).

Αντικαθιστώντας  $V_{CC} = 5V$ ,  $V_{CEsat} = 0.4V$  και  $I_C = 0.8mA$  στην ανίσωση (5.2.11) προκύπτει ότι η αντίσταση φόρτου πρέπει να είναι  $R_L > 5.75\text{k}\Omega$ . Σημειώνεται ότι για την αλλαγή της εξόδου του ενισχυτή χρησιμοποιείται τιμή χαμηλότερη της  $I_{CH}$  ώστε για ένταση ρεύματος από την τιμή αυτήν και πάνω να παράγεται η υψηλή έξοδος του ενισχυτή.

Η διάμετρος του άξονα στο ύψος του αισθητήρα είναι 0.42in. Επομένως, η περιφέρεια άξονα και, συνεπώς το μήκος της ταινίας κωδικοποίησης είναι

$$C = \pi d \approx 3.35\text{cm} \quad (5.2.12)$$

Διάστημα εναλλαγής 4mm σε αυτό το μήκος ταινίας δημιουργεί  $\frac{C}{X_d} \approx \frac{3.35}{0.4}$   $\frac{\text{cm}}{\text{cm}} \approx 8.37$  τμήματα ταινίας. Το πλήθος των τμημάτων που καλύπτουν την περιφέρεια του άξονα πρέπει να είναι άρτιος αριθμός ώστε να αποτελούν μία συνέχεια καθώς ο άξονας ολοκληρώνει μία πλήρη περιστροφή και ξεκινάει την επόμενη. Επομένως, τα τμήματα επιλέγονται να είναι 8 και βάσει αυτού, υπολογίζεται νέο διάστημα εναλλαγής  $X_d = \frac{C}{8} \approx 0.42\text{cm}$ .

Εφόσον η αντίσταση φόρτου έχει υπολογιστεί και είναι γνωστό και το πλήθος των εναλλαγών, ανάγεται η συχνότητα αποκοπής περίπου 6kHz ( $R_L = 6.9\text{k}\Omega$  για  $V_{CC} = 5\text{V}$ , σχήμα 5.2.5). Με κάθε πλήρη περιστροφή του άξονα, προκαλούνται 8 εναλλαγές ενώ επιτρέπονται κατά το μέγιστο 6000 εναλλαγές το δευτερόλεπτο. Επομένως, ένα μέγιστο υπολογίζεται στα  $\frac{6000}{8} \cdot \frac{\frac{\text{εναλλ}}{\text{sec}}}{\frac{\text{εναλλ}}{\text{rev}}} = 750\text{rps}$ , υπερβολικά υψηλό για τις απαιτήσεις της υλοποίησης.



## Κεφάλαιο 6

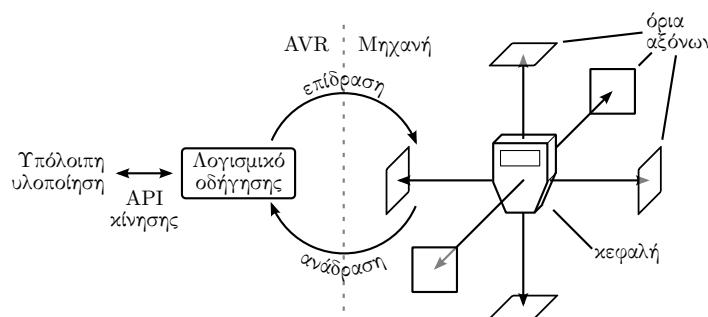
### Τυποσύστημα κίνησης

Στο κεφάλαιο Κατασκευή (σ. 49) αναλύθηκαν τα συστατικά στοιχεία και οι επιμέρους διατάξεις που συνθέτουν τη φυσική υπόσταση της συσκευής, μέρος του οποίου είναι αφερωμένο στους άξονες κίνησης. Όπως αναφέρεται σε εκείνο το κεφάλαιο, οι μετρήσεις πραγματοποιούνται από ένα κινητό εξάρτημα της συσκευής, την κεφαλή, που μετακινείται στο χώρο που ορίζουν αυτοί οι τρεις, στον αριθμό, άξονες.

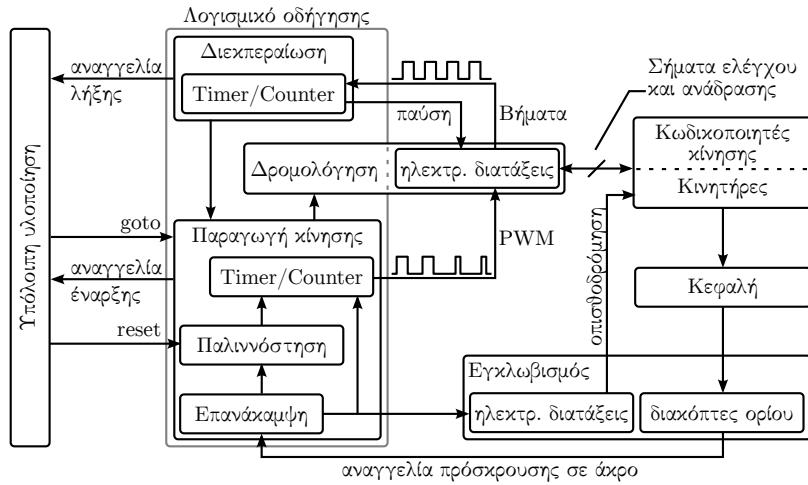
Στο παρόν κεφάλαιο αναλύεται η διασύνδεση του μικροελεγκτή με τους κινητήρες και συμπληρωματικές ηλεκτρονικές και μηχανικές διατάξεις για την ανάπτυξη ενός συστήματος ελέγχου της μετατόπισης της κεφαλής (βλ. σχήμα 6.0.1).

Τρεις σερβοκινητήρες συνεχούς περιστροφικής κίνησης (εφεξής, κινητήρες) – ένας για κάθε άξονα – χρησιμοποιούνται για την παραγωγή κίνησης με γωνιακή ταχύτητα που ελέγχεται μέσω ενός σήματος από παλμούς κυμαινόμενου πλάτους, την αναφερόμενη ως διαμόρφωση PWM (Pulse-Width Modulation). Μέσω της χρήσης μηχανών, η περιστροφική κίνηση μετατρέπεται σε γραμμική κίνηση της

Σχήμα 6.0.1: Τυποσύστημα κίνησης.



Σχήμα 6.0.2: Ανάλυση υποσυστήματος κίνησης.



κεφαλής. Τα υπόλοιπα συστήματα της υλοποίησης αντιλαμβάνονται και χειρίζονται τη θέση της κεφαλής μέσω συντεταγμένων τριών συνιστωσών (X, Y και Z) που χυμαίνονται εντός συγκεκριμένων ορίων (Συντεταγμένες και Λειτουργικό εύρος σ. 87). Αρχικά, αναλύονται τα διαθέσιμα κυκλώματα Χρονομετρητών/Απαριθμητών (Timer/Counter) του μικροελεγκτή που υποστηρίζουν παραγωγή σήματος PWM και συγχρίνονται τα χαρακτηριστικά τους σε σχέση με τις απαιτήσεις των κινητήρων και της υλοποίησης προκειμένου να επιλεγεί και διευθετηθεί το πλέον κατάλληλο (Παραγωγή κίνησης σ. 88).

Εν συνεχείᾳ, στην ενότητα Δρομολόγηση (σ. 92) αναφέρεται η βασική ανάγκη για πολύπλεξη ορισμένων γραμμών ελέγχου, όπως οι γραμμές σήματος PWM, ώστε να είναι δυνατό να υποστηριχθούν και οι τρεις κινητήρες μέσω του περιορισμένου αριθμού ακροδεκτών που δύνανται να παράγουν σήμα PWM. Στην ενότητα Διεκπεραίωση (σ. 93) περιγράφεται η ανάπτυξη ενός μηχανισμού που επιτρέπει το ίδιο το υλικό του μικροελεγκτή να σταμάτα την κίνηση των κινητήρων όταν ολοκληρώνεται μία προκαθορισμένη μετατόπιση, ώστε να εξασφαλίζεται ότι η κεφαλή ακινητοποιείται ακόμα και εάν τη στιγμή εκείνη, η CPU του μικροελεγκτή είναι απασχολημένη με άλλες εργασίες. Στην ίδια ενότητα παρουσιάζεται πώς, τελικά, επιτυγχάνεται ταυτόχρονη κίνηση της κεφαλής στο επίπεδο X-Y (Μέγιστη κοινή μετατόπιση σ. 95).

Το κεφάλαιο ολοκληρώνεται με την περιγραφή ορισμένων συμπληρωματικών μηχανισμών, του Εγκλωβισμού και επανάκαμψης (σ. 97) και της Παλινόστησης (σ. 101). Ο μηχανισμός του εγκλωβισμού προφυλάσσει τη συσκευή από την πρόκληση βλαβών, κυρίως στους κινητήρες, επιβάλλοντας την ακινητοποίησή της όταν

η κεφαλή προσχρούει στα όχρα της συσκευής. Λόγω της κρισιμότητας της λειτουργίας του, ο εγκλωβισμός υλοποιείται σε μηχανικό επίπεδο με χρήση ανασταλτικών διακοπών. Οι διακόπτες χρησιμοποιούνται ως ένασμα για το λογισμικό το οποίο, με τη σειρά του, προκαλεί την επαναφορά της κεφαλής σε μία γνωστή θέση (τη θέση επιστροφής) (βλ. Παλιννόστηση σ. 101). Επιπλέον περιγράφεται πώς η συσκευή αποπειράται εκ νέου, και για μία μόνο ακόμη φορά, τη μετακίνηση στη θέση που είχε προηγουμένως προκαλέσει εγκλωβισμό. Αυτό συμβαίνει ώστε να εξαλείφεται το ενδεχόμενο ότι ο εγκλωβισμός οφειλόταν σε αδυναμία παρακολούθησης της πραγματικής θέσης της κεφαλής (αστοχία υλικού) και που ίσως πλέον καταστεί δυνατό λόγω της πρόσφατης επαναφοράς της κεφαλής στη θέση επιστροφής (λειτουργία επανάκαμψης).

Η λογική συσχέτιση μεταξύ των μονάδων του υποσυστήματος κίνησης περιγράφεται από το σχήμα 6.0.2 ενώ η ανάλυσή τους πραγματοποιείται στις επόμενες ενότητες.

## 6.1 Συντεταγμένες και Λειτουργικό εύρος

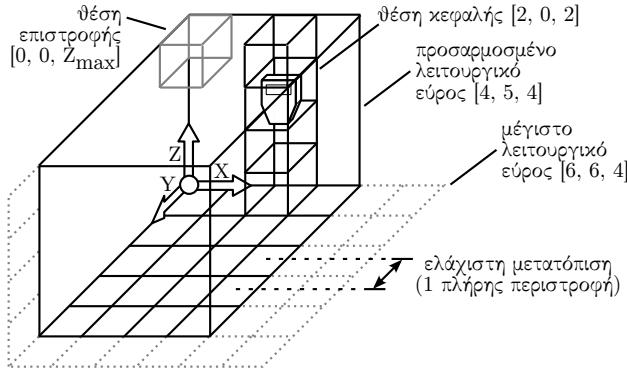
Τρεις άξονες γραμμικής κίνησης χρησιμοποιούνται για τη μετακίνηση της κεφαλής πάνω από την επιφάνεια του παρακολουθούμενου υλικού και κατακορύφως προς και από αυτό για την πραγματοποίηση μετρήσεων. Οι άξονες υποδιαιρούνται σε ένα πλήθος νοητών διακριτών θέσεων στις οποίες μπορεί να μεταβεί η κεφαλή. Το πλήθος αυτών των θέσεων εξαρτάται από την ελάχιστη δυνατή μετατόπιση που μπορεί να διατίθεται η κεφαλή καθώς και το συνολικά διαθέσιμο μήκος των αντίστοιχων ραγών της συσκευής (πάνω στις οποίες κινείται η κεφαλή). Στο πλαίσιο της υλοποίησης, η ελάχιστη μετατόπιση ορίζεται ως μία πλήρης περιστροφή του κινητήρα του αντίστοιχου άξονα. Πρακτικά, αυτό συνεπάγεται μετατοπίσεις των 4cm, περίπου. Επιπλέον, επιλέγεται να υποστηρίζεται ένα μέγιστο πλήθος 255 μετατοπίσεων ανά άξονα, δηλαδή ένα μέγιστο μήκος 1km ανά ράγα.

Σε κάθε περίπτωση, το διαθέσιμο πλήθος θέσεων της κεφαλής ορίζουν το αναφερόμενο ως λειτουργικό εύρος του οποίου η μέγιστη τιμή εξαρτάται άμεσα από τις διαστάσεις των ραγών της συσκευής. Ωστόσο, προκειμένου να παρέχεται μεγαλύτερη ευελιξία, το υποσύστημα κίνησης είναι δυνατό να διευθετείται δυναμικά μέσω λογισμικού ώστε να λειτουργεί σε ένα προσαρμοσμένο λειτουργικό εύρος, υποσύνολο του μέγιστου δυνατού (βλ. Πόρος /configuration σ. 134). Η δυνατότητα αυτή επιτρέπει την παρακολούθηση μικρότερων επιφανειών από τη μέγιστη υποστηριζόμενη. Το σχήμα

Την κάθε στιγμή, η θέση της κεφαλής προσδιορίζεται από τρεις τιμές· μία για τη θέση που κατέχει στον άξονα X, μία για τον άξονα Y και μία για το Z, που

### Σχήμα 6.1.1: Συντεταγμένες και Λειτουργικό εύρος.

Σημειώνεται ότι οι συνιστώσες των συντεταγμένων αφιύμονται από το 0, ενώ το λειτουργικό εύρος (το οποίο αντιπροσωπεύει μέγεθος διάστασης) από το 1.



συνολικά αναφέρονται ως οι συντεταγμένες (της θέσης) της κεφαλής. Προφανώς, η κεφαλή μπορεί να τεθεί μόνο σε συντεταγμένες που βρίσκονται εντός του τρέχοντος λειτουργικού εύρους (είτε αυτό είναι το μέγιστο δυνατό είτε προσαρμοσμένο).

## 6.2 Παραγωγή κίνησης

Σύμφωνα με εγχειρίδιο χρήσης της Hitec (2002), όλοι οι κινητήρες της λειτουργούν σε τάση 4.8–6V δεχόμενοι τετραγωνικό σήμα ελέγχου με διαμόρφωση PWM (Pulse-Width Modulation) 3–5V σε συχνότητα ανανέωσης 50Hz. Η διάρκεια των παλμών χυμαίνεται μεταξύ 0.9ms και 2.1ms, με παλμούς των 1.5ms να χρησιμοποιούνται για την ακινητοποίησή τους (Hitec, 2002).

Οι παραγωγή των παλμών μπορεί να πραγματοποιηθεί είτε με λογισμικό είτε από κάποιο διαθέσιμο κύκλωμα Χρονομετρητή/Απαριθμητή (Timer/Counter) του μικροελεγκτή. Το κύριο πλεονέκτημα του λογισμικού είναι η ελευθερία επιλογής οποιουδήποτε ακροδέκτη για την έξοδο του σήματος καθώς και η δυνατότητα παραγωγής πολλαπλών τέτοιων σημάτων που περιορίζονται από το συνολικό πλήθος ακροδεκτών ή τη συχνότητα του ρολογιού συστήματος (όποιο είναι μικρότερο).

Αντιθέτως, στην περίπτωση χρήσης κυκλωμάτων, το παραγόμενο σήμα διοχετεύεται στους συγκεκριμένους ακροδέκτες με τους οποίους είναι το καθένα εσωτερικά συνδεδεμένο και, σαφώς, το μέγιστο πλήθος παράλληλων σημάτων περιορίζεται από το συνολικό αριθμό τέτοιων κυκλωμάτων. Επίσης, κάθε Χρονομετρητής/Απαριθμητής υπόκειται σε πρόσθετους περιορισμούς, όπως το εύρος τιμών που υποστηρίζει ο καθένας (για παράδειγμα, ανάλυση 8- ή 16-bit).

Ωστόσο, στη χρήση κυκλωμάτος Χρονομετρητή/Απαριθμητή συγκαταλέγονται

και ορισμένα σημαντικά πλεονεκτήματα. Καθότι το σήμα παράγεται από ξεχωριστό κύκλωμα, η CPU του μικροελεγκτή είναι διαθέσιμη για την εκτέλεση οποιασδή- ποτε άλλης εργασίας (για παράδειγμα, κάποιας ρουτίνας εξυπηρέτησης διακοπής) παράλληλα με την παραγωγή του σήματος, χωρίς να απαιτείται πολύπλεξη εργασιών (ώστε να αποδίδονται κβάντα τόσο στη γεννήτρια σήματος όσο και σε κάποια άλλη εργασία). Ένα δεύτερο, λιγότερο σημαντικό για την προκειμένη υλοποίηση, πλεονεκτήμα είναι η δυνατότητα επίτευξης υψηλότερων συχνοτήτων με πολύ λιγότερο θόρυβο (jitter) και μεγαλύτερη ακρίβεια.

Τελικά, για την παραγωγή σήματος PWM των κινητήρων επιλέγεται η χρήση κυκλώματος Χρονομετρητή/Απαριθμητή καθώς, επιπροσθέτως των ανωτέρω πλεονεκτήμάτων, παρουσιάζει υψηλότερο ενδιαφέρον, από εκπαιδευτικής πλευράς.

### 6.2.1 Γεννήτρια PWM

Ο μικροελεγκτής ATmega328P διαθέτει δύο κυκλώματα Χρονομετρητή/Απαριθμητή με δυνατότητα παραγωγής σήματος PWM. Ο πρώτος (Timer/Counter0) είναι 8-bit και ο δεύτερος (Timer/Counter1), 16-bit. Για την επιλογή κάποιου, κρίνεται σκόπιμη η περαιτέρω ανάλυση της λειτουργίας της γεννήτριας PWM καθώς και των απαιτήσεων για το σήμα ελέγχου των κινητήρων.

Όσον αφορά τη γεννήτρια PWM, ο Χρονομετρητής/Απαριθμητής διαθέτει έναν καταχωρητή μετρητή, τον TCNTn, ο οποίος σταδιακά αυξάνεται. Επίσης, διαθέτει δύο ακόμα καταχωρητές, τους OCRnA και OCRnB, οι οποίοι αντιστοιχίζονται με ακροδέκτες του μικροελεγκτή, αναφερόμενοι ως OCnA και OCnB. Μόλις η τιμή του TCNTn γίνει ίση με την τιμή που περιέχεται είτε στον OCRnA είτε στον OCRnB, είναι δυνατή η αλλαγή της εξόδου του αντίστοιχου ακροδέκτη. Ο TCNTn συνεχίζει την ανοδική του πορεία έως ότου φτάσει την τιμή TOP. Από το σημείο αυτό και ανάλογα με τη λειτουργία που έχει επιλεγεί, ο TCNTn είτε επανέρχεται ακαριαία στην τιμή 0 και αρχίζει εκ νέου τον επόμενο κύκλο, είτε φθίνει σταδιακά μέχρι το 0, πάντα εναλλάσσοντας την τιμή των OCnA και OCnB όποτε εξισώνεται με τους OCRnA και OCRnB, αντιστοίχως (Atmel, 2013, σσ. 100–102, 124–129). Γίνεται αντιληπτό ότι η συχνότητα με την οποία αυξάνεται ο TCNTn καθώς και η τιμή TOP επηρεάζουν τη συχνότητα του παραγόμενου σήματος. Παράλληλα, η τιμή των OCRnA και OCRnB επηρεάζουν τον κύκλο εργασίας του (duty cycle), δηλαδή το τμήμα της περιόδου κατά το οποίο το παραγόμενο σήμα βρίσκεται σε λογικό 1.

Η παραγωγή PWM υποστηρίζεται από τρεις ρυθμίσεις λειτουργίας των Χρονομετρητών/Απαριθμητών, τις Fast PWM mode, Phase Correct PWM mode (PCPWM) και Phase and Frequency Correct PWM mode (PFCPWM), εκ των οποίων οι δύο τελευταίες ενδείκνυνται για τον έλεγχο κινητήρων εν γένει, και αυτό

επειδή το παραγόμενο σήμα ανταποκρίνεται πιο ομαλά στις αλλαγές της τιμής TOP (Atmel, 2013, σσ. 126,128). Οι PCPWM και PFCPWM, εφόσον η τιμή TOP – η τιμή μέχρι την οποία αυξάνει ο TCCnT πριν αρχίσει τη φθίνουσα πορεία του – διατηρείται σταθερή κατά τη διάρκεια λειτουργίας του μετρητή, είναι πανομοιότυπες (Atmel, 2013, σ. 127). Στην περίπτωση της υλοποίησης, η συχνότητα του παραγόμενου σήματος είναι σταθερή, όπως έχει αναφερθεί, στα 50Hz και, συνεπώς, το ίδιο ισχύει για την τιμή TOP. Ως αποτέλεσμα, οι λειτουργίες PCPWM και PFCPWM είναι ισοδύναμες για τις ανάγκες της υλοποίησης και μπορεί να προτιμηθεί είτε η μία είτε η άλλη, στην περίπτωση που επιλεγεί ο Timer/Counter1, ή η PCPWM, στην περίπτωση που επιλεγεί ο Timer/Counter0, καθώς σύμφωνα με τις επιλογές παραγωγής κυματομορφής της Atmel (2013, σ. 107), ο Timer/Counter0 υποστηρίζει μόνο αυτήν.

### Συχνότητα και κύκλος εργασίας

Στις λειτουργίες PCPWM και PFCPWM, η συχνότητα του παραγόμενου σήματος παρέχεται από τη σχέση (Atmel, 2013, σσ. 102,128,129):

$$f_{PWM} = \frac{f_{clk_{I/O}}}{2 N TOP} \quad (6.2.1)$$

όπου,

$f_{clk_{I/O}}$  : Συχνότητα ρολογιού που λαμβάνουν οι μετρητές (καθώς και άλλα περιφερειακά, όπως SPI).

$N$  : Τιμή υποδιαίρεσης ρολογιού για χρήση από τους μετρητές (1, 8, 64, 256 ή 1024).

$TOP$  : Η μέγιστη τιμή που παίρνει ο TCCnT.

Η συχνότητα ρολογιού,  $f_{clk_{I/O}}$ , της υλοποίησης ορίζεται στα 4MHz, ενώ έχει ήδη αναφερθεί η επιμυητή συχνότητα του παραγόμενου σήματος,  $f_{PWM}$ , (50Hz). Η τιμή υποδιαίρεσης,  $N$ , προσδιορίζεται από τα bit CSn2:0 του καταχωρητή TCCRnB, και επιτρέπει τη μείωση της συχνότητας των μετρητών. Η χρήση της τιμής  $TOP$  έχει αναφερθεί προηγουμένως. Η τιμή της προσδιορίζεται κατά την επιλογή της λειτουργίας παραγωγής κυματομορφής μέσω των bit WGM των καταχωρητών TCCRnA και TCCRnB. Στον πίνακα 6.2.1 παρουσιάζονται ορισμένες ρυθμίσεις PCPWM των δύο μετρητών. Σημειώνεται ότι στην περίπτωση του Timer/Counter0, οι δύο λειτουργίες που αναφέρονται είναι οι μοναδικές PCPWM, ενώ στην περίπτωση του Timer/Counter1, έχουν παραλειψεί ορισμένες οι οποίες είναι παρόμοιες με την λειτουργία 1 του Timer/Counter0 (δηλαδή, προκαθορισμένες σταθερές τιμές 0x00FF, 0x01FF και 0x3FF).

Οι λειτουργίες που παρέχουν μία προκαθορισμένη σταθερή τιμή (0xFF/0x00FF, 0x01FF και 0x3FF) απορρίπτονται καθώς, εάν αντικατασταθούν στη σχέση (6.2.1),

Πίνακας 6.2.1: Μέρος ρυθμίσεων PCPWM των Timer/Counter0 και 1.

	Mode	WGMn3	WGMn2	WGMn1	WGMn0	TOP
Timer/Counter0	1	–	0	0	1	0xFF
	5	–	1	0	1	OCR0A
Timer/Counter1	10	1	0	1	0	ICR1
	11	1	0	1	1	OCR1A

Απόσπασμα Atmel Corporation (2013). *ATmega48A, ATmega48PA, ATmega88A, ATmega88PA, ATmega168A, ATmega168PA, ATmega328, ATmega328P datasheet*. Έκδ. 1.7. 660 σσ. url: <http://www.atmel.com/devices/ATMEGA328.aspx?tab=documents> (επίσκεψη 01/01/2014), σσ. 107,133

αποτυγχάνουν να παράξουν μία αποδεκτή τιμή υποδιαίρεσης,  $N$ , και, κατ' επέκταση, συχνότητα 50Hz. Οι υπόλοιπες λειτουργίες, οι οποίες χρησιμοποιούν ως τιμή TOP την τιμή που τίθεται στον αναφερόμενο, κάθισ φορά, καταχωρητή, παρέχουν περισσότερη ευελιξία και ακρίβεια καθώς επιτρέπουν την αυθαίρετη επιλογή μίας αποδεκτής τιμής  $N$  και βάσει αυτής να υπολογιστεί η τιμή TOP.

Ωστόσο, στην περίπτωση του Timer/Counter0, ο καταχωρητής στον οποίο εκχωρείται η τιμή TOP μπορεί μόνο να είναι ο OCR0A. Συνεπώς, μόνο ο ακροδέκτης OC0B μπορεί να χρησιμοποιηθεί ως έξοδος σήματος PWM, ο κύκλος εργασίας του οποίου ελέγχεται από τον OCR0B. Εάν επιλεγεί αυτή η διευθέτηση του κυκλώματος θα είναι δυνατό να ελεγχθεί η κίνηση ενός μόνο κινητήρα τη φορά, κάτι που αποτρέπει την παράλληλη κίνηση σε δύο άξονες. Συνεπώς, προτιμάται η λειτουργία 10 του Timer/Counter1, η οποία επιτρέπει να αποστέλλεται σήμα PWM διαφορετικού κύκλου εργασίας δια μέσω του ενός ή και των δύο ακροδέκτων.

### Διευθέτηση γεννήτριας PWM

Είναι γνωστό ότι η επιλογή της τιμής υποδιαίρεσης,  $N$ , μπορεί να γίνει πλέον αυθαίρετα. Στην πραγματικότητα, εφαρμόζοντας τις διαθέσιμες τιμές που αναφέρει η Atmel (2013, σ. 109) – 1, 8, 64, 256 και 1024 – στη σχέση (6.2.1), μόνο οι τρεις πρώτες παράγουν ακέραιο αριθμό. Ωστόσο, ποια από τις τρεις είναι κατάλληλη;

Στο σημείο αυτό κρίνεται απαραίτητη η λεπτομερέστερη εξέταση των χαρακτηριστικών των κινητήρων σχετικά με το αναμενόμενο σήμα ελέγχου. Η συχνότητα σήματος 50Hz συνεπάγεται περιόδους των 20ms, εκ των οποίων, το σήμα ελέγχου βρίσκεται σε λογικό 1 για 0.9–2.1ms (Hitec, 2002). Για απλούστευση, θεωρείται εύρος παλμών 1–2ms, καθώς οι ακραίες τιμές παράγουν τη μέγιστη γωνιακή ταχύτητα που, ούτως ή άλλως, ξεπερνά τις απαιτήσεις της υλοποίησης. Συνεπώς, ο κύκλος εργασίας του παραγόμενου σήματος πρέπει να κυμαίνεται μόλις στο 5–10%. Στον πίνακα 6.2.2 υπολογίζονται, βάσει της σχέσης (6.2.1) και για τις τρεις πρώτες διαθέσιμες τιμές υποδιαίρεσης,  $N$ , η τιμή TOP που παράγει 50Hz συχνότητα

Πίνακας 6.2.2: Κύκλοι εργασίας και τιμές OCR1x.

Υποδιαιρεση	TOP	5%	7.5%	10%
1	40000	2000	3000	4000
8	5000	250	375	500
64	625	32	46-47	62

σήματος. Οι στήλες «5%», «7.5%» και «10%» περιέχουν την τιμή των OCR1x που παράγουν (ή πλησιάζουν, στην περίπτωση όπου  $N = 64$ ) τον αντίστοιχο κύκλο εργασίας. Οι κύκλοι εργασίας κοντά στο 7.5% χρησιμοποιούνται για την ακινητοποίηση του κινητήρα, ενώ οι τιμές μέχρι τα δύο άκρα (5% και 10%), αυξάνουν σταδιακά τη γωνιακή ταχύτατα (αριστερόστροφα και δεξιόστροφα, αντιστοίχως).

Φαινομενικά, τα αποτελέσματα προτρέπουν τη χρήση υποδιαιρεσης  $N = 1$  ώστε να παρέχεται μεγαλύτερος έλεγχος του κύκλου εργασίας του σήματος. Ωστόσο, στην πραγματικότητα, οι ερασιτεχνικοί κινητήρες που χρησιμοποιούνται στην υλοποίηση, ενδεχομένως οι κινητήρες εν γένει, αδυνατούν να διαχρίνουν τόσο μικρές διαφοροποίησεις στο σήμα, που σημαίνει ότι παρότι το εύρος τιμών είναι μεγαλύτερο, οι περισσότερες τιμές αλληλοκαλύπτονται και προκαλούν την ίδια συμπεριφορά κινητήρα. Για παράδειγμα, με  $TOP = 3001$ , ο κύκλος εργασίας είναι 7.5025%, ενώ με  $TOP = 3010$ , 7.525%. Και στις δύο περιπτώσεις – προφανώς, και σε όλες τις ενδιάμεσες – ο κινητήρας παραμένει ακίνητος. Παρόμοια συμπεριφορά παρουσιάζεται και για μεγαλύτερες γωνιακές ταχύτητες.

Στο αντίθετο άκρο, η χρήση υποδιαιρεσης  $N = 64$  παρέχει πολύ λίγο έλεγχο στη διαχύμανση της ταχύτητας· μόλις 14 διαχριτές τιμές για κάθε φορά περιστροφής. Στον πλαίσιο της υλοποίησης επιλέγεται υποδιαιρεση  $N = 8$  και οι αντίστοιχες ρυθμίσεις.

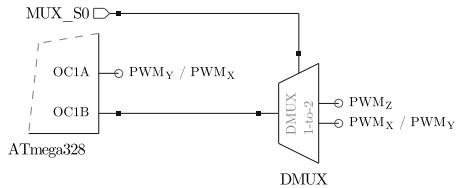
### 6.3 Δρομολόγηση σημάτων

Ο μετρητής Timer/Counter1 διευθεύτεται ώστε να παράγονται μέχρι δύο σήματα PWM, ανεξαρτήτου κύκλου εργασίας το καθένα αλλά της ίδιας συχνότητας. Συνεπώς, είναι δυνατός ο ταυτόχρονος χειρισμός μέχρι και δύο κινητήρων. Ωστόσο, εάν είναι αποδεκτός ο χειρισμός περισσότερων, του ενός, κινητήρων από την ίδια γραμμή ελέγχου με ενεργοποίηση μόνο του ενός κάθε φορά, τότε μπορούν να υποστηριχθούν περισσότεροι, με χρήση κάποιου εξωτερικού κυκλώματος επιλογής του επιλυμητού, όπως για παράδειγμα, ενός αποπολυλέκτη.

Η υλοποίηση χρησιμοποιεί τρεις κινητήρες, έναν για κάθε άξονα κίνησης, οι οποίοι συμβολικά ονομάζονται, ανάλογα με τον άξονα που εξυπηρετούν, κινητήρας X, κινητήρας Y και κινητήρας Z. Οι κινητήρες X και Y μετακινούν την κεφαλή σε

Σχήμα 6.3.1: Δρομολόγηση σήματος στους κινητήρες.

Παρότι απεικονίζεται αντιστοίχιση του αποπολυπλέκτη με τον ακροδέκτη OC1B, όταν μπορούσε, κάλλιστα, να είχε γίνει με τον OC1A. Ο MUX\_S0 είναι ένας οποιοσδήποτε ελεύθερος ακροδέκτης του μικροελεγκτή.



επίπεδο παράλληλο του παρακολουθούμενου υλικού, ενώ ο κινητήρας Z, κατακόρυφα προς το υλικό, ώστε οι αισθητήρες να διεισδύουν σε κατάλληλο βάθος πριν τη λήψη μετρήσεων.

Λόγω της φύσεως του υλικού και για την μείωση αντιστάσεων κατά την κίνηση αλλά και την αποφυγή πιθανών φυσικών της κεφαλής ή και των αισθητήρων, εφαρμόζεται ολική ανύψωση της κεφαλής προτού πραγματοποιηθεί μετακίνηση σε νέα θέση στο επίπεδο X-Y. Αντιστοίχως, η κεφαλή χαμηλώνεται μόνο όταν οι άλλοι δύο κινητήρες βρίσκονται σε ηρεμία. Συνεπώς, προκύπτει ότι οι κινητήρες X και Y μπορούν να είναι ενεργοποιημένοι ταυτόχρονα, ο ένας λαμβάνοντας σήμα από τον ακροδέκτη OC1A και ο άλλος, από τον OC1B. Ωστόσο, για να συμπεριληφθεί και ο κινητήρας Z, το σήμα του καθώς και το σήμα ενός εκ των άλλων δύο κινητήρων αποδίδεται πότε στον έναν και πότε στον άλλο μέσω αποπολυπλέκτη. Το σχήμα 6.3.1 παρουσιάζει αυτήν την παραδοχή.

### 6.3.1 Διεκπεραίωση

Κάθε κινητήρας που χρησιμοποιείται στην υλοποίηση είναι άρρηκτα συνδεδεμένος με έναν οπτικό κωδικοποιητή περιστροφικής κίνησης (εφεξής, κωδικοποιητής), που επιτρέπει την παρακολούθηση της κίνησης του. (Οι κωδικοποιητές αναλύονται σε βάθος στo [REF].) Ο κωδικοποιητής διαθέτει ένα σήμα εξόδου από το οποίο αποστέλλονται διαδοχικοί παλμοί για κάθε βήμα του κινητήρα. Καταμετρώντας τους παλμούς είναι δυνατό να αναχθεί η μετατόπιση που έχει προκληθεί σε κάθε άξονα ώστε, τελικά, να ακινητοποιηθούν οι κινητήρες, εφόσον η κεφαλή έχει φτάσει στην επιθυμητή θέση.

Σαφώς, η καταμέτρηση είναι δυνατό να πραγματοποιείται από το λογισμικό του μικροελεγκτή. Ωστόσο, για τους ίδιους λόγους που επιλέγεται παραγωγή σήματος PWM από το υλικό του μικροελεγκτή και όχι από το λογισμικό, μελετάται το ενδεχόμενο μίας αντίστοιχης πορείας.

### Μηχανική διεκπεραίωση

Μία πιθανή λύση παρέχεται από το Χρονομετρητή/Απαριθμητή Timer/Counter0, ο οποίος υποστηρίζει τη χρήση εξωτερικού ρολογιού για την αύξηση του καταχωρητή/μετρητή, TCNT0, σε κάθε εισερχόμενο παλμό στον ακροδέκτη T0 (Atmel, 2013, σ. 109). Εφόσον ενεργοποιηθεί μέσω του καταχωρητή TIMSK0, μόλις η τιμή του TCNT0 γίνει ίση με την τιμή του επιλεγμένου καταχωρητή σύγκρισης, OCR0x, προκαλείται διακοπή και εκτελείται η αντίστοιχη ρουτίνα εξυπηρέτησης, γνωστοποιώντας, με αυτόν τον τρόπο, το συμβάν (Atmel, 2013, σ. 110). Επομένως, εάν, πριν την ενεργοποίηση κάποιου κινητήρα, είναι δυνατό να υπολογιστεί το πλήθος το βημάτων που απαιτείται για την μετάβαση στην επιθυμητή θέση, τότε η τιμή αυτή μπορεί να τεθεί στον OCR0x και, κατόπιν, να ενεργοποιηθεί ο μετρητής. Έτσι, είναι βέβαιο ότι το λογισμικό θα ειδοποιηθεί όταν τα βήματα έχουν ολοκληρωθεί.

Ενδέχεται, ωστόσο, τη στιγμή που τίθεται ο ενδείκτης OCF0x του καταχωρητή TIFR0 και που, φυσιολογικά, θα εκτελούται η αντίστοιχη ρουτίνα εξυπηρέτησης, οι διακοπές να έχουν απενεργοποιηθεί επειδή βρίσκεται ήδη σε εξέλιξη η εκτέλεση μίας άλλης ρουτίνας. Σε αυτήν την περίπτωση, μετά την ολοκλήρωση της τρέχουσας ρουτίνας και την επαναφορά των διακοπών, θα δούμε στις εν αναμονή διακοπές η ευκαιρία να εξυπηρετηθούν, πάντα με σειρά προτεραιότητας (Atmel, 2013, σσ. 13–14,57). Το ενδέχομενο αυτό θα μπορούσε να προκαλέσει ανεπιθύμητες παρενέργειες, ιδίως εάν η καθυστέρηση είναι παρατεταμένη. Για παράδειγμα, η κεφαλή θα συνέχιζε την πορεία της για όσο χρόνο αυτή υφίσταται και όταν, τελικά, η διακοπή θα εξυπηρετούται, η κεφαλή θα βρίσκεται σε θέση διαφορετική από την αναμενόμενη και χωρίς να υπάρχει κάποια σχετική ένδειξη.

Για την αποφυγή τέτοιων καταστάσεων, κρίνεται απαραίτητη η ύπαρξη ενός μηχανισμού, σε επίπεδο υλικού, που να πυροδοτείται με την ολοκλήρωση των βημάτων και, με τη σειρά του, να απενεργοποιεί τον κινητήρα. Θα μπορούσε να αντιπροσωπεύει την αλλαγή της κατάστασης από *off* σε *on*, ενός διακόπτη που παρεμβάλλεται της πηγής του σήματος PWM και του κινητήρα ώστε να εμποδίζει την μετάδοση του σήματος.

Η λύση παρέχεται, εν μέρει, από τον ίδιο τον μετρητή με τη δυνατότητά του για παραγωγή εξόδου στον ακροδέκτη OC0A, όταν ο TCNT0 εξισώνεται με τον OCR0A (Atmel, 2013, σσ. 98–99,107). Η λειτουργία ονομάζεται CTC (Clear Timer on Compare Match) και η παραγόμενη έξοδος επηρεάζεται από τα bit COM0A1:0 του καταχωρητή TCCR0A. Από τις τρεις ρυθμίσεις (Toggle, Clear και Set) επιλέγεται η Toggle σύμφωνα με την οποία η έξοδος εναλλάσσεται σε κάθε εξίσωση των TCNT0 και OCR0A. Ωστόσο, αν ο διακόπτης είναι active high, και προκειμένου να τίθεται σε κατάσταση off που επιτρέπει τη μετάδοση του

σήματος, η τιμή του OC0A πρέπει, αρχικά, να είναι λογικό 1. Η επιβολή αυτής της τιμής γίνεται με χρήση του bit FOC0A του καταχωρητή TCCR0B η οποία αλλάζει την έξοδο στον ακροδέκτη OC0A σαν να είχε προκύψει εξίσωση μεταξύ TCNT0 και OCR0A χωρίς, ωστόσο, να προκαλείται διακοπή ή επανέναρξη του TCNT0.

Επομένως, τη στιγμή που λαμβάνεται το τελευταίο βήμα από τον κωδικοποιητή, η έξοδος του OC0A αλλάζει σε λογικό 0 και, έτσι, αποτρέπει την αναμετάδοση του σήματος PWM (ακροδέκτες OC1A και OC1B), παρότι η γεννήτρια PWM συνεχίζει να λειτουργεί. Όποτε η CPU του μικροελεγκτή γίνει διαθέσιμη, ενημερώνεται για το συμβάν (καθώς εκτελείται η σχετική ρουτίνα εξυπηρέτησης) και από εκεί είτε απενεργοποιείται η γεννήτρια PWM είτε δρομολογείται νέα μετακίνηση.

Ο παραπάνω σχεδιασμός, παρότι επιτυγχάνει ένα κρίσιμο χαρακτηριστικό της υλοποίησης – τη μηχανική διακοπή των κινητήρων – επιβάλλει ορισμένα σημεία επανεξέτασης. Αρχικά, καθώς ο μετρητής Timer/Counter0 διαθέτει καταχωρητές των 8-bit (δηλαδή, καθένας από τους TCNT0, OCR0A και OCR0B είναι 8-bit), η μέγιστη τιμή TOP είναι το 255. Αυτό συνεπάγεται ότι κάθε μεμονωμένη μετακίνηση διαθέτει, το πολύ, 257 βήματα. Εάν απαιτούνται περισσότερα βήματα, τότε η συνολική μετατόπιση πρέπει να διασπάται σε μικρότερες μετακινήσεις. (Σημειώνεται ότι τα βήματα είναι, το πολύ, 257 και όχι 256, καθώς 256 παλμοί απαιτούνται για την μετάβαση του TCNT0 από 0 σε 255. Ωστόσο, η εξίσωση ενεργοποιείται στον επόμενο παλμό – τον 257 – ο οποίος και θέτει τον TCNT0 σε 0.)

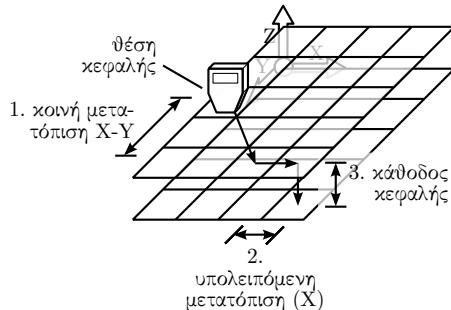
Για λόγους πληρότητας, αναφέρεται, επίσης, ότι η συχνότητα των εξωτερικών παλμών υπόκειται σε έναν βασικό περιορισμό. Σύμφωνα με το εγχειρίδιο της Atmel (2013, σσ. 139–140), η συχνότητα του εξωτερικού ρολογιού πρέπει να είναι μικρότερη από  $f_{clk_{I/O}}/2.5$ , ώστε να είναι εγγυημένη η αναγνώριση όλων των παλμών. Ωστόσο, στην περίπτωσή της υλοποίησης, οι εισερχόμενοι παλμοί είναι, μόλις, της τάξης των μερικών δεκάδων Hz, ενώ το ρολόι συστήματος, των μερικών MHz.

Το σημαντικό σημείο προς μελέτη, ωστόσο, είναι το πλήθος των μετρητών βημάτων που δύνανται να διευθετηθούν. Εφόσον ο Timer/Counter1, ο οποίος, επίσης, υποστηρίζει συγχρονισμό με εξωτερικό ρολόι, διευθετείται για την παραγγή σήματος PWM, απομένει μόνο ο Timer/Counter0 για αυτήν τη λειτουργία και, προφανώς, για έναν μόνο κωδικοποιητή τη φορά. Όπως έχει αναφερθεί, είναι επιμυητή η ταυτόχρονη κίνηση στους άξονες X και Y, όποτε αυτό απαιτείται.

### Μέγιστη κοινή μετατόπιση

Κρίνεται αναγκαίος και ικανός ο ακόλουθος ανασχεδιασμός. Ο ακροδέκτης T0 λαμβάνει το βήμα ενός εκ των τριών κωδικοποιητών κίνησης, με την επιλογή κάποιου να γίνεται μέσω πολυπλέκτη. Στην περίπτωση όπου απαιτείται κίνηση και στους δύο άξονες X και Y, ο Timer/Counter0 διευθετείται ώστε να καταμετράει

Σχήμα 6.3.2: Αλληλουχία κύκλων μετατόπισης.



τα βήματα ενός εκ των δύο. Οι δύο κινητήρες δροιμολογούνται για τη μέγιστη κοινή, κατά μέτρο, μετατόπιση. Με την ολοκλήρωση του πρώτου κύκλου βημάτων, η πλήρης μετατόπιση τουλάχιστον του ενός εκ των δύο αξόνων θα έχει καλυφθεί. Εφόσον εκχρεμεί μετατόπιση για τον άλλο, ξεκινάει ένας δεύτερος κύκλος για τα υπολειπόμενα βήματα. Η αλληλουχία των κύκλων μετατόπισης πασουριάζεται στο σχήμα 6.3.2.

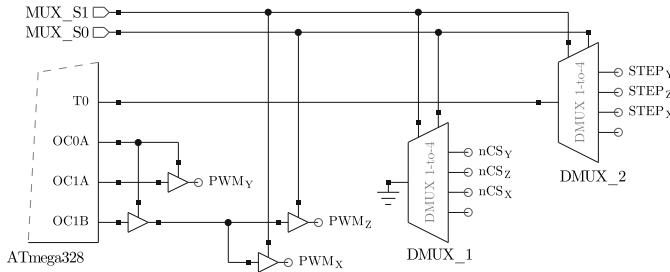
Αυτός ο σχεδιασμός επιβάλλει, ωστόσο, οι κινητήρες X και Y, να λειτουργούν με την ίδια γωνιακή ταχύτητα, ώστε στο χρόνο που απαιτεί ο ένας να περιστραφεί ένα συγκεκριμένο αριθμό βημάτων, να ολοκληρώνει και ο άλλος τον ίδιο αριθμό βημάτων. Στην υλοποίηση, και κατόπιν αρχετών αναπροσαρμογών του κύκλου εργασίας των κινητήρων, η συμπεριφορά αυτή προσεγγίζεται σε βαθμό που παράγει, σχετικά, ικανοποιητικά αποτελέσματα.

Στο σχήμα 6.3.3 εμφανίζεται η ανασχεδιασμένη παραδοχή όπου διακόπτες ελεγχόμενοι από την έξοδο του μετρητή βημάτων παρεμβάλλονται των γραμμών του σήματος PWM, ενώ ο αποπολυπλέκτης 2 τροφοδοτεί το μετρητή με βήματα ενός εκ των τριών καδικοποιητών. Ο αποπολυπλέκτης 1-προς-2 του προηγούμενου σχεδιασμού έχει αντικατασταθεί από δύο διακόπτες οι οποίοι παρέχουν ισοδύναμα αποτελέσματα, δεδομένου ότι οι καδικοποιητές συνδέονται στα τρία πρώτα κανάλια του αποπολυπλέκτη και ότι η παραγωγή σήματος PWM είναι απενεργοποιημένη όταν η λογική τιμή των MUX<sub>S1:0</sub> είναι 0x3. Η δεύτερη προϋπόθεση είναι αυτονόητη καθώς θεωρείται μη έγκυρη κατάσταση η προώθηση σήματος σε κινητήρα, χωρίς την ενεργοποίηση του αντίστοιχου καδικοποιητή.

Η χρήση διακοπών αντί του 1-προς-2 πολυπλέκτη έχει το επιπρόσθετο πλεονέκτημα ότι μαζί με τους άλλους δύο διακόπτες σχηματίζεται ένα τετραμερές ολοκληρωμένο κύκλωμα διακοπών όπως, για παράδειγμα, το CD4016.

Επίσης, έχει εισαχθεί ένας ακόμα αποπολυπλέκτης ο οποίος θέτει σε λειτουργία τον αντίστοιχο καδικοποιητή, ουσιαστικά, ενεργοποιώντας τη δίοδο υπερύθρων

Σχήμα 6.3.3: Λογική σύνδεση μετρητή βημάτων και κινητήρων.



που διαθέτει. Για λόγους που έχουν αναφερθεί σε προηγούμενο κεφάλαιο, όπως η επιμήκυνση της διάρκειας ζωής της διόδου καθώς και η μείωση της κατανάλωσης ισχύος, ο κάθε κωδικοποιητής ενεργοποιείται μόνο για το χρονικό διάστημα που περιστρέφεται ο αντίστοιχος κινητήρας.

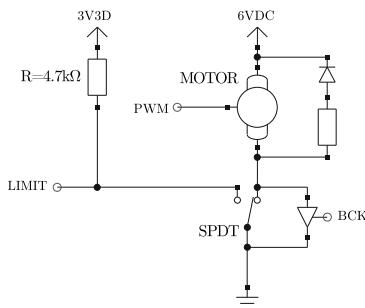
Τα εναπομένατα κανάλια των δύο αποπολυπλεκτών μπορούν να χρησιμοποιηθούν για την κάλυψη άλλων αναγκών, δεδομένου ότι η χρήση γίνεται όταν το υποσύστημα κινητήρων βρίσκεται σε αδράνεια. Για παράδειγμα, το τέταρτο κανάλι του MUX\_1 μπορεί να συνδεθεί με τον ακροδέκτη  $\overline{CS}$  κάποιου άλλου ολοκληρωμένου, ενώ του MUX\_2, για την ανταλλαγή δεδομένων όταν ο μετρητής βημάτων είναι απενεργοποιημένος.

## 6.4 Εγκλωβισμός και επανάκαμψη

Η ύπαρξη των κωδικοποιητών κίνησης αιτιολογείται από την ανάγκη ανατροφοδότησης σχετικά με την πορεία εξέλιξης της μετατόπισης σε κάθε άξονα. Ωστόσο, οι κωδικοποιητές, αυτοσχέδιοι ή μη, είναι, ένα βαθύμ, επιρρεπείς σε εσφαλμένες μετρήσεις, κάτι που καθορίζεται από τη διακριτική τους ικανότητα (Albert, 2011, σσ. 15–16). Επιπλέον, η υπόθεση ότι, κατά τη μέγιστη κοινή μετατόπιση, ο αριθμός βημάτων που πραγματοποιείται σε κάθε άξονα είναι ο ίδιος και για τους δύο, εισάγει επιπρόσθετη πιθανότητα αστοχίας. Ανεξαρτήτως της συχνότητας εμφάνισης, κρίνεται απαραίτητη η ύπαρξη ενός μηχανισμού ως έσχατη προστασία κατά της εξώθησης της κεφαλής πέραν των φυσικών ορίων (του πλαισίου) της συσκευής που οφείλεται σε αδυναμία αναγνώρισης ή παρακολούθησης της πραγματικής θέσης της κεφαλής.

Η λύση περιλαμβάνει τη στερέωση μηχανικών διακοπών σε κάθε πλευρά των κινητών μερών της συσκευής ώστε κατά την πρόσκρουσή τους με τα άκρα του πλαισίου, αφενός να απενεργοποιούνται οι κινητήρες και, αφετέρου, να ειδοποιείται ο μικροελεγκτής για το συμβάν. Και σε αυτήν την περίπτωση, θεωρείται ζωτικής

Σχήμα 6.4.1: Συνδεσμολογία εγκλωβισμού και επανάκαμψης κινητήρα σε ένα άκρο.



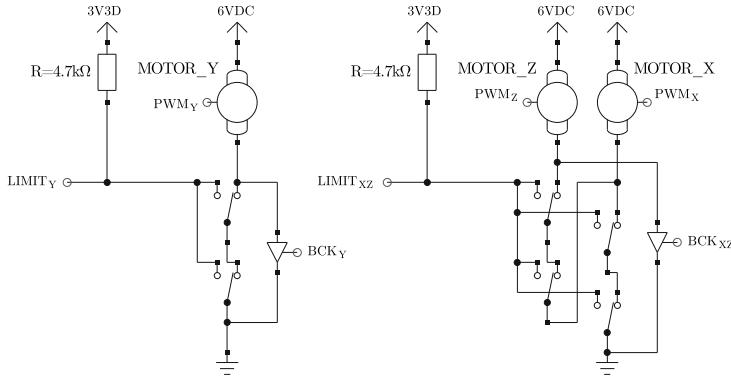
σημασίας η απενεργοποίηση να συμβαίνει σε επίπεδο υλικού, ενώ το λογισμικό να επεμβαίνει στην πορεία, όταν ευχαρίστησε. Το σχήμα 6.4.1 παρουσιάζει τη βασική ιδέα.

Ο επιλεγμένος διακόπτης είναι SPDT (Single Pole, Double Throw) και, συνεπώς, διαιθέτει τρεις ακροδέκτες· όταν ο διακόπτης είναι ελεύθερος, ο ακροδέκτης NC (Normally Closed) συνδέεται με τον COM (Common), ενώ όσο βρίσκεται πιεσμένος, ο ακροδέκτης NO (Normally Open) συνδέεται με τον COM, ενώ ο NC είναι αποσυνδεδεμένος. Η τοποθέτηση του διακόπτη στη γραμμή τροφοδοσίας του κινητήρα επιτρέπει τη μηχανική αποσύνδεσή του όποτε ο διακόπτης πιέζεται.

Προκειμένου να ειδοποιείται ο μικροελεγκτής ότι ο κινητήρας έχει ακινητοποιηθεί, όταν πρέπει η αλλαγή της κατάστασης του διακόπτη να προκαλεί εναλλαγή σε κάποιο εισερχόμενό του σήμα. Δεδομένου ότι η σύνδεση του ακροδέκτη NO με τον COM θέτει το σήμα αυτό σε λογικό 0, το σήμα όταν πρέπει, υπό φυσιολογικές συνθήκες, να βρίσκεται σε λογικό 1. Αυτό αιτιολογεί την ύπαρξη αντιστάτη pull-up στο σχήμα 6.4.1, ώστε η γραμμή LIMIT να τίθεται σε λογικό 1, από προεπιλογή. Από πλευράς ρυθμίσεων, ο μικροελεγκτής αρκεί να διευθετηθεί ώστε να προκαλείται διακοπή στην εναλλαγή του σήματος του συνδεδεμένου ακροδέκτη (level triggered ή edge triggered). Οι λεπτομέρειες αναλύονται στην Αναγγελία εγκλωβισμού (σ. 100).

Η αναγνώριση του συμβάντος από το μικροελεγκτή είναι απαραίτητη ώστε ο κινητήρας να τίθεται σε κατάλληλη θέση για τη συνέχιση της λειτουργίας του. Αφενός απαιτείται η αλλαγή του κύκλου εργασίας του σήματος PWM ώστε ο κινητήρας να περιστρέψεται με φορά αντίθετη αυτής που προκάλεσε τη διακοπή για χρονικό διάστημα μέχρι την απελευθέρωση του διακόπτη. Ωστόσο, η αλλαγή του κύκλου εργασίας από μόνη της είναι αδύνατο να επαναφέρει τον κινητήρα καθώς αυτός είναι μηχανικά αποσυνδεδεμένος από την τροφοδοσία (εξαιτίας του πιεσμένου διακόπτη). Για το λόγο αυτό, παρέχεται ένας ηλεκτρονικός διακόπτης ως εφεδρική

Σχήμα 6.4.2: Πλήρης συνδεσμολογία εγκλωβισμού και επανάχαμψης.



σύνδεση με την τάση αναφοράς (GND). Όταν ο κινητήρας έχει οπισθοδρομήσει αρκετά ώστε να απεγκλωβιστεί από τον μηχανικό διακόπτη, ο ηλεκτρονικός διακόπτης απενεργοποιείται και ο κινητήρας είναι σε θέση να λειτουργηθεί κανονικά.

Στο σχήμα απεικονίζεται, επίσης, μία δίοδος επιστροφής συνδεδεμένη στα άκρα του κινητήρα. Ο λόγος ύπαρξής της είναι η εξομάλυνση παλμών που δημιουργούνται κατά την κατάρρευση του μαγνητικού πεδίου του πηνίου του κινητήρα όταν αυτός αποσυνδέεται από την τροφοδοσία (Kuphaldt, 2009, σσ. 130–132). Ωστόσο, χρίνεται ότι στην περίπτωση της υλοποίησης, η οποία χρησιμοποιεί σερβοκινητήρες, η ανάγκη για τη συμπερίληψή τους είναι μικρή και, τελικά, αποκλείονται από αυτήν.

#### 6.4.1 Προσαρμογή στις ανάγκες της υλοποίησης

Η προαναφερθείσα συνδεσμολογία αγνοεί το γεγονός ότι απαιτούνται δύο ανασταλτικοί διακόπτες SPDT ανά κινητήρα – έναν για κάθισμα άκρο – καθώς και το γεγονός ότι ο κινητήρας Z κινείται μόνο εφόσον οι άλλοι δύο βρίσκονται σε ηρεμία. Το τελευταίο θα μπορούσε να χρησιμοποιηθεί για τη μείωση του αριθμού των παραγόμενων σημάτων. Στο σχήμα 6.4.2 παρουσιάζεται η τελική συνδεσμολογία. Παρατηρείται ότι οι ανασταλτικοί διακόπτες των αξόνων X και Z συνδέονται σε σειρά και ότι χρησιμοποιείται μόνο ένας ηλεκτρονικός διακόπτης για την εφεδρική σύνδεση με την τάση αναφοράς (BCK<sub>XZ</sub>). Αντιστοίχως, δεσμεύεται μόνο ένας ακροδέκτης του μικροελεγκτή για την ειδοποίηση εγκλωβισμού κάποιου εκ των δύο κινητήρων (LIMIT<sub>XZ</sub>).

Ο λόγος για τη συγχώνευση των κυκλωμάτων των κινητήρων X και Z αντί των Y και Z είναι η διευκόλυνση της φυσικής διασύνδεσης των διακοπών λόγω της εγγύτητας των δύο αξόνων. Η σύνδεση X και Y, σαφώς, απορρίπτεται, καθώς

αυτοί οι δύο κινητήρες δύνανται να τεθούν σε λειτουργία ταυτόχρονα και θα ήταν, συνεπώς, αδύνατο να εντοπιστεί ποιος από τους δύο έχει εγκλωβιστεί.

Το βασικό μειονέκτημα της επιλεγμένης μεθόδου είναι ότι επιτρέπει την αναγνώριση του κυκλώματος κινητήρων και όχι του συγκεκριμένου διακόπτη που προκαλεί τον εγκλωβισμό. Συνεπώς, εφόσον οι κινητήρες τίθενται σε κίνηση από το μικροελεγκτή και στην πορεία προκύπτει εγκλωβισμός, η κατάσταση είναι δυνατό να αναστραφεί. Ωστόσο, αν κατά την εκκίνηση της συσκευής κάποιος κινητήρας είναι εγκλωβισμένος, αυτός ο μηχανισμός από μόνος του είναι ανεπαρκής για την επανάκαμψή του. Η παρούσα υλοποίηση αρκείται σε αυτόν το μηχανισμό χωρίς να καταβάλει προσπάθεια επανάκαμψης στην προαναφερθείσα περίπτωση.

#### 6.4.2 Αναγγελία εγκλωβισμού

Αναφέρθηκε προηγουμένως η ανάγκη διευθέτησης του μικροελεγκτή για την απόκριση του στην εναλλαγή των σημάτων LIMIT<sub>XZ</sub> και LIMIT<sub>Y</sub>. Σύμφωνα με το εγχειρίδιο του μικροελεγκτή της Atmel (2013, σ. 71), εξωτερικές διακοπές είναι δυνατό να ενεργοποιηθούν για εισερχόμενες παρυφές (edge triggered) σε οποιοδήποτε ακροδέκτη PCINT ή, επιπροσθέτως, για οποιαδήποτε λογική αλλαγή στους ακροδέκτες INT0 και INT1. Προτιμάται η χρήση δύο PCINT ακροδεκτών καθώς καλύπτουν τις απαιτήσεις ενώ, ταυτόχρονα, επιτρέπουν τη χρήση των ειδικών ακροδεκτών INT0 και INT1 σε περιπτώσεις όπου διακοπή σε εναλλαγή σήματος είναι ακατάλληλη.

Το σύνολο των ακροδεκτών PCINT διαμοιράζεται σε τρεις ομάδες. Η εναλλαγή του σήματος ενός οποιουδήποτε ακροδέκτη κάθισε ομάδας προκαλεί την εκτέλεση της ίδιας ρουτίνας εξυπηρέτησης, εφόσον το bit της αντίστοιχης ομάδας του καταχωριητή PCICR (Pin Change Interrupt Control Register) έχει τεθεί. Επιπλέον, με τους καταχωρητές PCMSK (Pin Change Mask Register) – 1, 2 και 3 – προσδιορίζονται ποιοι PCINT ακροδέκτες κάθισε ομάδας είναι επιθυμητό να προκαλούν διακοπή.

Το γεγονός ότι όλοι οι ακροδέκτες του μικροελεγκτή αντιστοιχίζονται με κάπιον PCINT, καθιστούν εύκολη την επιλογή και ανάθεση δύο εξ αυτών στα σήματα LIMIT<sub>XZ</sub> και LIMIT<sub>Y</sub>. Ωστόσο, προτιμώνται δύο ακροδέκτες PCINT που ανήκουν στην ίδια ομάδα με αποτέλεσμα να καλείται η ίδια ρουτίνα εξυπηρέτησης ώστε η αναγνώριση του εγκλωβισμένου κινητήρα να επαφίεται στο ίδιο το λογισμικό (σαφώς με τη βοήθεια των σημάτων).

## 6.5 Παλιννόστηση

Κατά την ενεργοποίηση της συσκευής (σύνδεση με τροφοδοσία) καθώς και κατά τον εγκλωβισμό και επανάκαμψη κάποιου κινητήρα, χρίνεται απαραίτητη η επαναφορά της κεφαλής σε μία γνωστή θέση, τη θέση επιστροφής (machine home) (Albert, 2011, σ. 99). Από εκείνο το σημείο, είναι δυνατόν η κεφαλή να δρομολογηθεί σε νέα θέση ή και σε αυτήν που προκάλεσε τον εγκλωβισμό.

Κατά βάση, ο μηχανισμός είναι ίδιος και στις δύο περιπτώσεις. Αρχικά, ο κινητήρας Z διευθετείται για την πραγματοποίηση του μέγιστου δυνατού – και όχι το μέγιστου αποδεκτού – πλήθους βημάτων με διεύθυνση που οδηγεί την κεφαλή στη θέση επιστροφής για αυτόν τον άξονα. Εν τέλει, προκαλείται, νέος εγκλωβισμός κινητήρα. Και σε αυτήν την περίπτωση, ο μηχανισμός επανάκαμψης αναλαμβάνει, αυτόματα, την οπισθοδρόμηση της κεφαλής έως ότου απεγκλωβιστεί ο κινητήρας. Ωστόσο, με την ολοκλήρωση της επανάκαμψης, είναι γνωστό ότι η κεφαλή βρίσκεται στη θέση επιστροφής για τον άξονα Z. Σε δεύτερο στάδιο, εφαρμόζεται η ίδια διαδικασία, αυτήν τη φορά, για τους άξονες X και Y, ταυτόχρονα. Με την επανάκαμψη και των δύο, η κεφαλή θεωρείται ότι βρίσκεται στη θέση επιστροφής (συντεταγμένες [0, 0, Z<sub>max</sub>]).

Αναφέρθηκε, προηγουμένως, η δυνατότητα της συσκευής για την εκ νέου δρομολόγηση της κεφαλής σε περίπτωση κατά την οποία η προηγούμενη διακόπτεται απρόοπτα από εγκλωβισμό κινητήρα. Ο λόγος είναι ότι το συμβάν του εγκλωβισμού μπορεί να οφείλεται σε αστοχία παρακολούθησης της θέσης της κεφαλής. Σε αυτήν την περίπτωση, και εφόσον η κεφαλή έχει μόλις αρχικοποιηθεί, ενδέχεται η νέα προσπάθεια να επιφέρει τα επιθυμητά αποτελέσματα. Η υπόθεση βασίζεται στην παραδοχή ότι κάθιμε μετακίνηση εισάγει ένα ποσοστό απόκλισης μεταξύ αναγνωρισμένης και πραγματικής θέση της κεφαλής. Επομένως, πολλαπλές μετακινήσεις ενδέχεται να προκαλέσουν μεγαλύτερη συνολική απόκλιση. Κατόπιν ολοκλήρωσης της παλιννόστησης, η συσκευή διαθέτει μία πρόσφατη και έγκυρη αναγνώριση θέσης.

Ωστόσο, σε περίπτωση που η επιθυμητή θέση είναι, στην πραγματικότητα, απροσπέλαστη, για παράδειγμα, εξαιτίας κάποιου παρεμβαλλόμενου εμποδίου, η παραπάνω συμπεριφορά προκαλεί έναν ατέρμονα βρόχο παλιννόστησης και νέας δρομολόγησης. Η κατάσταση αυτή, προφανώς ανεπιθύμητη, αποφεύγεται αποτρέποντας τη δρομολόγηση της ίδιας θέσης εφόσον, τη στιγμή του εγκλωβισμού, υπάρχει ένδειξη ότι είναι η πρώτη μετακίνηση μετά από παλιννόστηση.



## Κεφάλαιο 7

### Διαδικτύωση

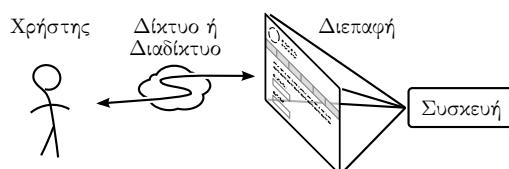
Η επικοινωνία με τη συσκευή επιλέγεται να πραγματοποιείται μέσω Διαδικτύου και, για την ακρίβεια, κάνοντας χρήση του βασικού μηχανισμού του Παγκοσμίου Ιστού, το πρωτόκολλο HTTP (HyperText Transfer Protocol). Πρωταρχικός λόγος είναι η μεγάλη διάδοσή του με όμεσο αποτέλεσμα την πληθώρα συσκευών και της ευκολίας με την οποία πραγματοποιείται η πρόσβαση.

Προκειμένου να επιτευχθεί αυτό, η συσκευή αποδίδεται δικτυακή υπόσταση ώστε να είναι δυνατή η πρόσβασή της σε επίπεδο τοπικού δικτύου του χώρου όπου στεγάζεται (τυπικά, μέσω IEEE 802.3 10BASE-T ή IEEE 802.3u 100BASE-TX) και μέσω αυτού, και υποστηρίζοντας τα απαραίτητα πρωτόκολλα, την επέκτασή της στον Παγκόσμιο Ιστό (βλ. σχήμα 7.0.2).

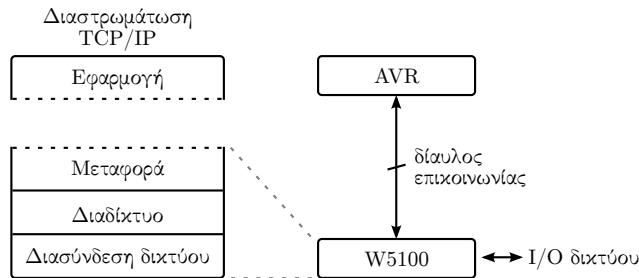
Μέρος της εργασίας αναλαμβάνεται από ανεξάρτητο ολοκληρωμένο κύκλωμα, το W5100 της WIZnet, το οποίο παρέχει τα τρία χαμηλότερα επίπεδα της στοιβασίας TCP/IP – Διασύνδεση δικτύου, Διαδίκτυο και Μεταφορά – επιτρέποντας την υλοποίηση να παράσχει μόνο το επίπεδο της Εφαρμογής, το οποίο, στην προκει-

**Σχήμα 7.0.1:** Επικοινωνία χρήστη και συσκευής μέσω πρωτοκόλλων διαδικτύωσης.

Ο χρήστης αναπαριστά την εκάστοτε εξωτερική οντότητα που επιθυμεί να επικοινωνήσει με τη συσκευή και ενδέχεται να είναι είτε άνθρωπος είτε υπολογιστικό σύστημα. Η διεπαφή είναι το μέσο (ή γλώσσα) της μεταξύ τους επικοινωνίας η οποία, προφανώς, πρέπει να είναι αποδεκτή και γνωστή και από τους δύο συμμετέχοντες.



Σχήμα 7.0.2: Διαμοιρασμός αρμοδιοτήτων μεταξύ μικροελεγκτή και W5100.



μένη, αποτελεί έναν διακομιστή (server) HTTP (βλ. σχήμα 7.0.2). Στην ενότητα Το ολοκληρωμένο δικτύωσης W5100 (σ. 105) αναλύεται πώς επιτυγχάνεται η επικοινωνία μεταξύ μικροελεγκτή και ολοκληρωμένου (Διασύνδεση με μικροελεγκτή σ. 107), οι απαραίτητες ρυθμίσεις των καταχωρητών του για τη λειτουργία του TCP Socket του διακομιστή, καθώς και οι προδιαγραφές του λογισμικού οδήγησης που επιτρέπει το γενικότερο χειρισμό του και την ανταλλαγή δεδομένων (Διευθέτηση W5100 σ. 110).

Στο Συμπληρωματικό λογισμικό W5100 (σ. 115) περιγράφεται κάποιο επιπρόσθετο λογισμικό που χρίνεται απαραίτητο για τη διευκόλυνση επεξεργασίας κειμένου από το εκάστοτε Socket πάνω στο οποίο στηρίζονται υψηλότερα λογικά επίπεδα του διακομιστή.

Στην πορεία, αναλύεται το λογισμικό του διακομιστή (Τυποσύστημα διακομιστή HTTP σ. 119), το οποίο είναι υπεύθυνο για τη λήψη, επεξεργασία και απόκριση αιτημάτων HTTP. Δια μέσω των αιτημάτων-αποχρίσεων συντίθεται μία διεπαφή που επιτρέπει την πρόσβαση και τον έλεγχο της συσκευής τόσο μέσω λογισμικού πλοιογγησης (browser) με φιλική, προς το χρήστη, αναπαράσταση (ιστοσελίδα), όσο και μέσω τρίτων πληροφοριακών συστημάτων με αυτοματοποιημένο τρόπο.

Αρχικά, περιγράφεται η βασική συμπεριφορά και οι προδιαγραφές του διακομιστή, ο οποίος αναπτύσσεται σύμφωνα με το πρωτόκολλο HTTP/1.1 (Κορμός διακομιστή σ. 120). Για παράδειγμα, όλες οι υλοποιήσεις HTTP/1.1 πρέπει να υποστηρίζουν τη λήψη τεμαχισμένων (chunked) μηνυμάτων. Στο πλαίσιο της υλοποίησης, αναπτύσσεται μηχανισμός που επιτρέπει την ανασύνθεση τέτοιων μηνυμάτων χωρίς αυτό να γίνεται αντιληπτό από λογισμικό υψηλότερου λογικού επιπέδου. Επιπλέον, παρέχονται εργαλεία για την παραγωγή τεμαχισμένων μηνυμάτων καθώς και ενός κοινού τρόπου σύνταξης των πεδίων κεφαλίδας της απόκρισης.

Μία επόμενη βασική μονάδα είναι ο Αναλυτής HTTP (σ. 124) του οποίου έργο είναι η αναγνώριση των στοιχείων της κεφαλίδας του αιτήματος, με πλέον σημαντικά, τη μέθοδο, το URI, τις παραμέτρους ερωτήματος (query string σ. 128)

και ορισμένα πεδία της που ενδιαφέρουν την υλοποίηση. Η πληροφορία που εξάγεται από τον Αναλυτή HTTP παραδίδεται σε ειδικό λογισμικό που είναι υπεύθυνο για την παραγωγή της απόχρισης, τις αναφέρομενες ως ρουτίνες περάτωσης. Η σχετική υποδομή περιγράφεται στους Πόρους διακομιστή (σ. 126).

Ένα τελευταίο κομμάτι του διακομιστή που περιγράφεται είναι πώς αναλύονται και δημιουργούνται αναπαραστάσεις πόρων. Ορισμένες αναπαραστάσεις δημιουργούνται σε προγενέστερο χρόνο και αποθηκεύονται στατικά σε κάποια δευτερεύουσα μνήμη της συσκευής (βλ. Πόροι αρχείων σ. 139). Ωστόσο, ορισμένες άλλες, επιβάλλεται να δημιουργούνται δυναμικά βάσει των στοιχείων του εκάστοτε αιτήματος (για παράδειγμα, ποιες είναι οι τελευταίες N καταχωρημένες μετρήσεις). Σε τέτοιες περιπτώσεις, χρησιμοποιείται η Μονάδα αναπαράστασης πόρων (σ. 129) για τη μετατροπή κάποιων δομών/μεταβλητών από τη μνήμη του μικροελεγκτή σε κατάλληλη αναπαράσταση. Η ίδια μονάδα χρησιμοποιείται και για την αντίστροφη λειτουργία που είναι η μετατροπή μίας αναπαράστασης πόρου που παρέχεται εντός αιτήματος σε μορφή που μπορεί να αξιοποιηθεί από τον μικροελεγκτή (μεταβλητές).

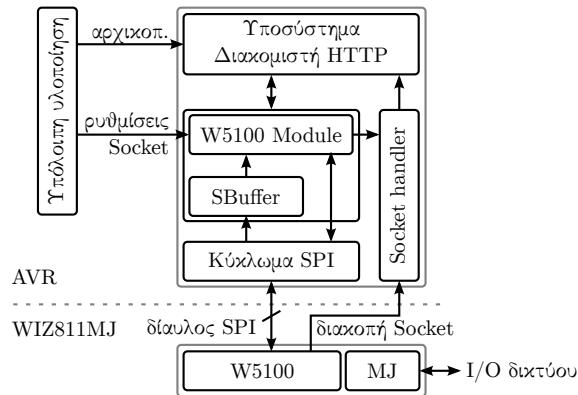
Τελικά, αναφέρονται οι διαθέσιμοι πόροι που ορίζονται στο πλαίσιο της υλοποίησης κάνοντας χρήση του Υποσυστήματος διακομιστή HTTP και που, ουσιαστικά, πρόκειται για το τι είναι δυνατό να πραγματοποιηθεί μέσω της διεπαφής της συσκευής. Αναφορικά, υποστηρίζεται η ρύθμιση της συσκευής (για παράδειγμα, διεύθυνση IP, χρόνος αναμονής και πλήθος μετρήσεων, λειτουργικό εύρος) (Πόρος /configuration σ. 134), η άμεση μετακίνηση της κεφαλής σε νέα θέση (Πόρος /coordinates σ. 136) καθώς και η προτροπή της για τη λήψη μίας νέας μέτρησης με μη αυτόματο τρόπο ή την ανάκτηση όλων ή μέρους των καταγεγραμμένων μετρήσεων (Πόρος /measurement σ. 137). Επιπλέον παρέχεται ο πόρος /index (ισοδύναμος του /) που, μαζί με ορισμένους άλλους (/style.css, /client.js και /logo.png), συνθέτουν μία υποτυπώδη ιστοσελίδα. Εφαρμόζοντας τεχνικές AJAX και κάνοντας χρήση ορισμένων πόρων του διακομιστή, η ιστοσελίδα επιτρέπει στο χρήστη το χειρισμό της συσκευής μέσω λογισμικού πλοήγησης.

## 7.1 Το ολοκληρωμένο δικτύωσης W5100

Η διασύνδεση της συσκευής με το δίκτυο επιλέγεται να γίνει με το ολοκληρωμένο κύκλωμα W5100 της WIZnet, το οποίο παρέχει τα υμελιώδη πρωτόκολλα της στοίβας TCP/IP μέχρι και το επίπεδο μεταφοράς, υλοποιημένα σε υλικό (για παράδειγμα, IEEE 802.3, IP, TCP, UDP) (WIZnet, 2011, σσ. 4–5). Στο πλαίσιο της υλοποίησης, μελετάται η διασύνδεση και ο χειρισμός του W5100 από το μικροελεγκτή και με αυτό ως βάση, χτίζεται ο διακομιστής HTTP.

Καθώς το ολοκληρωμένο μπορεί να λειτουργεί μέχρι τέσσερα, ταυτοχρόνως

Σχήμα 7.0.3: Μέρη που απαρτίζουν τη μονάδα διαδικτύωσης.



ενεργά, Socket ρυθμιζόμενου μεγέθους μνήμης εισερχόμενων και εξερχόμενων δεδομένων, προκύπτουν ορισμένες ιδιαιτερότητες στον τρόπο με τον οποίο γίνεται η ανταλλαγή τους με το μικροελεγκτή (λεπτομέρειες στη Διαχείριση μνήμης Socket σ. 112). Για το λόγο αυτό, καθώς και για τη γενικότερη διευκόλυνση των εργασιών μεταξύ μικροελεγκτή και ολοκληρωμένου, αναπτύσσεται, στο πλαίσιο της υλοποίησης, λογισμικό οδήγησης το οποίο, στη συνέχεια, χρησιμοποιείται ως βάση για την ανέγερση του λογισμικού επιπέδου εφαρμογής, του διακομιστή HTTP.

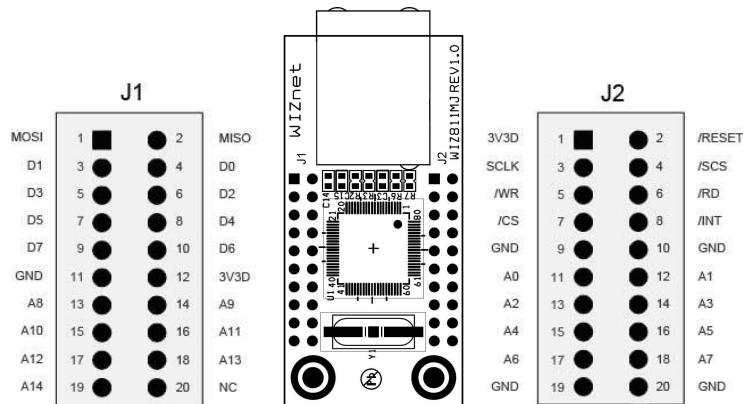
### 7.1.1 Πλακέτα WIZ811MJ

Η χρήση του ολοκληρωμένου επιλέγεται να γίνει δια μέσω της πλακέτας WIZ811MJ της ίδιας εταιρείας. Οι λόγοι είναι ότι επιλύει τη διασύνδεση του ολοκληρωμένου με συνδετήρα RJ-45 (απαραίτητος για την ανταλλαγή δεδομένων μεταξύ W5100 και δικτύου), ταλαντωτή και διάφορα άλλα ηλεκτρονικά στοιχεία. Από τους 80, συνολικά, ακροδέκτες του W5100, το WIZ811MJ διαθέτει μόλις 40, οι οποίοι είναι αυτοί που προορίζονται για τη διασύνδεση με μικροελεγκτή. Όλοι οι υπόλοιποι συμμετέχουν σε εσωτερικές συνδέσεις της πλακέτας.

Ένα παράδειγμα σχετικά με το τελευταίο, το W5100 διαθέτει έναν ακροδέκτη για το χειρισμό του ολοκληρωμένου ως Slave σε δίαυλο SPI, το  $\overline{\text{SCS}}$  (SPI Chip-Select) (WIZnet, 2011, σ. 9). Ωστόσο, διαθέτει και έναν επιπρόσθετο ακροδέκτη, τον SEN (SPI Enable), για την ενεργοποίηση των υποσυστημάτων του ολοκληρωμένου για λειτουργία σε SPI (WIZnet, 2011, σ. 8). Το WIZ811MJ παρέχει ακροδέκτη μόνο για το  $\overline{\text{SCS}}$  ενώ διαθέτει κατάλληλη διάταξη ώστε η αλλαγή της τιμής του να προκαλεί το αναμενόμενο σήμα στο SEN, αυτομάτως (WIZnet, 2013, σ. 7).

Επιπλέον σημαντικό χαρακτηριστικό είναι ότι η πλακέτα παρέχει ακροδέκτες

Σχήμα 7.1.1: Πλακέτα WIZ811MJ και διάταξη των ακροδεκτών του W5100.



*Pin assignments.* Στο WIZnet. *WIZ811MJ Datasheet*. Έχδ. 1.2. 8 Ιούλ. 2013. 13 σσ. url: [http://www.wiznet.co.kr/Sub\\_Modules/en/product/product\\_detail.asp?Refid=39&page=1&cate1=&cate2=&cate3=&pid=1030&cType=2#tab](http://www.wiznet.co.kr/Sub_Modules/en/product/product_detail.asp?Refid=39&page=1&cate1=&cate2=&cate3=&pid=1030&cType=2#tab) (επίσκεψη 25/01/2014), σ. 6

συμβατούς με πρωτότυπες κάρτες (breadboard) που χρησιμοποιεί η υλοποίηση, εν αντιθέσει με το W5100 το οποίο είναι SMD (Surface-Mount Device) και απαιτεί συγκόλληση (WIZnet, 2013, σσ. 6,12).

### 7.1.2 Διασύνδεση με μικροελεγκτή

Το W5100 υποστηρίζει τρεις τρόπους για την επικοινωνία με το μικροελεγκτή· άμεση ή έμμεση προσπέλαση ή, μέσω πρωτοκόλλου SPI (WIZnet, 2011, σ. 59). Οι δύο πρώτες μέθοδοι χρησιμοποιούν διαύλους διεύθυνσης και δεδομένων απαιτώντας πολλά σημεία σύνδεσης με το μικροελεγκτή· για την ακρίβεια, 3 γραμμές ελέγχου (Chip-Select, Read, Write), 8 γραμμές για το δίστημα δεδομένων, και, 15 ή 2 γραμμές διεύθυνσης για άμεση ή έμμεση προσπέλαση, αντίστοιχα. Στην περίπτωση του SPI απαιτούνται πολύ λιγότερες (μόλις 4) και αυτός είναι ο λόγος που προτιμάται για τη διασύνδεση του με το μικροελεγκτή, δεδομένου του περιορισμένου αριθμού ακροδεκτών του. Σαφώς, το μειονέκτημα χρήσης SPI – ενός σειριακού πρωτοκόλλου επικοινωνίας – είναι ότι επιτυγχάνεται πολύ μικρότερος ρυθμός ανταλλαγής δεδομένων από ότι στην περίπτωση των άλλων δύο, κάτι που, τελικά, επηρεάζει το χρόνο απόκρισης στα εισερχόμενα αιτήματα. Ωστόσο, κρίνεται ότι για τις ανάγκες της υλοποίησης, αυτός ο περιορισμός είναι αμελητέος σε σχέση με την εξοικονόμηση ακροδεκτών που αποφέρει η χρήση SPI.

### Συνδεσμολογία

Το σχήμα 7.1.1 παρουσιάζει την πλακέτα WIZ811MJ και, σε μεγέθυνση, την ονομασία των ακροδεκτών της. Από τους ακροδέκτες J1, ιδιαίτερου ενδιαφέροντος είναι οι 1 (MOSI), 2 (MISO) και 12 (3V3D) εκ των οποίων οι δύο πρώτοι χρησιμοποιούνται για την μετάδοση των bit του Master και του ενεργού Slave του διαύλου SPI, αντίστοιχα, ενώ ο τελευταίος, για την τροφοδοσία της πλακέτας με τάση 3.3V. Οι υπόλοιποι, με εξαίρεση τον 20 (Not Connected), συνδέονται με την τάση αναφοράς, δεδομένου ότι όλοι, με εξαίρεση τον ακροδέκτη 11 (GND), χρησιμοποιούνται μόνο στην περίπτωση άμεσης ή έμμεσης προσπέλασης.

Από τους ακροδέκτες J2, ο 1 (3V3D) χρησιμοποιείται για τροφοδοσία 3.3V ενώ οι 9, 10, 19 και 20 (GND) συνδέονται με την τάση αναφοράς. Οι 3 (SCLK) και ( $\overline{SCS}$ ) αποτελούν μέρος του διαύλου SPI για τη μεταφορά του ρολογιού από το Master και την ενεργοποίηση του W5100 ως Slave. Σύμφωνα με το εγχειρίδιο της WIZnet (2013, σ. 8), ο ακροδέκτης 2 ( $\overline{RESET}$ ) προκαλεί την αρχικοποίηση όλων των καταχωρητών του W5100 στις προεπιλεγμένες τους τιμές και κρίνεται ότι αρκεί να συνδεθεί με το αντίστοιχο σήμα του μικροελεγκτή ώστε η αρχικοποίηση του W5100 να πραγματοποιείται μαζί με του μικροελεγκτή.

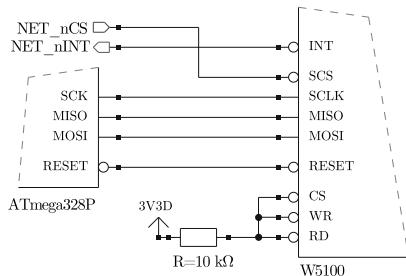
Οι ακροδέκτες 5 ( $\overline{WR}$ ), 6 ( $\overline{RD}$ ) και 7 ( $\overline{CS}$ ) χρησιμοποιούνται στην περίπτωση επικοινωνίας μέσω άμεσης ή έμμεσης προσπέλασης και όχι σε SPI (WIZnet, 2011, σ. 8). Ωστόσο, επειδή τα σήματα είναι active low, τίθενται σε μόνιμο λογικό 1 (3.3V), μέσω αντιστάτη, ώστε να είναι λογικά ανενεργά.

Μέσω του ακροδέκτη 8 ( $\overline{INT}$ ), και εφόσον έχει διευθετηθεί ακολούθως, το W5100 αναγγέλλει την επιθυμία του για επικοινωνία με το μικροελεγκτή (για παράδειγμα, επειδή έχουν καταφύγει δεδομένα). Σε αντίθετη περίπτωση, ο μικροελεγκτής είναι υποχρεωμένος να εξετάζει από μόνος του το ενδεχόμενο για επικοινωνία (polling). Για την υλοποίηση, κρίνεται αξιόλογη η χρήση των διακοπών. Περισσότερες λεπτομέρειες παρέχονται στην Κατάσταση και διακοπές (σ. 111).

Για λόγους που έχουν αναφερθεί, όλοι οι υπόλοιποι ακροδέκτες συνδέονται με την τάση αναφοράς.

Στο σχήμα 7.1.2 εμφανίζεται η συνδεσμολογία του μικροελεγκτή με τους ακροδέκτες του W5100 (δια μέσω της πλακέτα WIZ811MJ). Αξίζει να σημειωθεί ότι, παρόλο που το W5100 λειτουργεί με τάση έως και 3.6V, ανέχεται τάση στα σήματα εισόδου έως και 5.5V (5V tolerant) (WIZnet, 2011, σ. 64). Το χαρακτηριστικό αυτό είναι ιδιαίτερα σημαντικό επειδή καθιστά δυνατή τη διασύνδεσή του με το μικροελεγκτή χωρίς να απαιτούνται επιπρόσθετα ενδιάμεσα κυκλώματα για λογικό μετασχηματισμό.

Σχήμα 7.1.2: Διασύνδεση μικροελεγκτή και W5100.



Σχήμα 7.1.3: Δομή εντολής W5100 μέσω SPI.

op-code	address-H	address-L	payload
---------	-----------	-----------	---------

### Πρωτόκολλο επικοινωνίας

Το πρότυπο διαύλου επικοινωνίας SPI καθορίζει τα ηλεκτρικά χαρακτηριστικά για την επίτευξη αμφίδρομης ταυτόχρονης bit-προς-bit ζεύξης επικοινωνίας μεταξύ Master και Slave διατάξεων. Σε αυτήν την υποδομή στηρίζεται το πρωτόκολλο του W5100, το οποίο με απλές εγγραφές και αναγνώσεις των διευθύνσεων μνήμης επιτυγχάνει την επιθυμητή συμπεριφορά του W5100.

Στο σχήμα 7.1.3 εμφανίζεται η δομή των εντολών, οι οποίες συνίστανται από ένα Byte κωδικού (op-code), δύο Byte διεύθυνσης (address-H και address-L) και ένα Byte φορτίου (payload). Ο κωδικός αποστέλλεται πρώτος και προσδιορίζει εάν πρόκειται να εκτελεστεί εγγραφή (0xF0) ή ανάγνωση (0x0F) στη διεύθυνση μνήμης που καθορίζεται από τα επόμενα δύο Byte (με το πλέον σημαντικό Byte να αποστέλλεται πρώτο) (WIZnet, 2011, σ. 61). Στην περίπτωση όπου γίνεται ανάγνωση, η τιμή του φορτίου είναι αδιάφορη καθώς τα 8bit φορτίου αποστέλλονται ώστε να δοθεί η ευχαρίστια στο W5100 να επιστρέψει τα περιεχόμενα της αντίστοιχης διεύθυνσης (dummy data).

Πλέον, είναι εύληπτη η επιβάρυνση που επιφέρει η χρήση SPI στη συγκεκριμένη περίπτωση καθώς μεταφέρονται 32bit για, μόλις, 8bit πραγματικού φορτίου.

Επιπλέον, σύμφωνα με τις προδιαγραφές χρονισμού της WIZnet (2011, σ. 66), η μέγιστη υποστηριζόμενη συχνότητα ρολογιού υπολογίζεται περίπου στα 14MHz. Με τις τρέχουσες επιλογές της υλοποίησης, το ρολόι συστήματος ανέρχεται στα 4MHz με αποτέλεσμα η μέγιστη συχνότητα ρολογιού SPI που δύναται να παραχθεί από το μικροελεγκτή να είναι, σύμφωνα με το εγχειρίδιο της Atmel (2013, σσ. 179–180), τα 2MHz· κατά πολύ χαμηλότερη από το προαναφερθέν μέγιστο όριο.

### 7.1.3 Διευθέτηση W5100

Οι καταχωρητές του W5100 χωρίζονται σε δύο κύριες ομάδες· τους καταχωρητές γενικών ρυθμίσεων (ή κοινοί καταχωρητές) και τους καταχωρητές Socket. Οι κοινοί καταχωρητές επηρεάζουν τη συμπεριφορά του W5100 συνολικά ή προσδιορίζουν κάποια χαρακτηριστικά όλων των Socket, ενώ οι καταχωρητές Socket είναι υπεύθυνοι για τη λειτουργία καθενός εκ των τεσσάρων Socket του ολοκληρωμένου.

Όλοι οι καταχωρητές αντιστοιχίζονται μία διεύθυνση, ενώ υπάρχουν καταχωρητές περισσοτέρων του ενός byte. Η πρόσβαση σε αυτούς τους καταχωρητές, είτε πρόκειται για ανάγνωση είτε για εγγραφή, γίνεται από τη χαμηλότερη προς την υψηλότερη διεύθυνση με Big Endian διάταξη των byte (WIZnet, 2011, σσ. 32–33,35).

Οι κοινοί καταχωρητές τίθενται μόνο μία φορά, κατά την εκκίνηση της συσκευής, και, οι ορισμένοι, κατόπιν εισερχόμενων αιτημάτων (για παράδειγμα, αλλαγή διεύθυνσης IP). Σε αντίθεση, η πρόσβαση στους καταχωρητές Socket, οι οποίοι χρησιμοποιούνται για το χειρισμό κάθε Socket, είναι πιο συχνή.

#### Καταχωρητές διευθύνσεων και μάσκας υποδικτύου

Σε αυτήν την κατηγορία συγκαταλέγονται καταχωρητές που ρυθμίζουν τη διεπαφή για τη σύνδεση με το δίκτυο μέσω των GAR (Gateway Address Register), SUBR (Subnet Mask Register), SHAR (Source Hardware Address Register) και SIPR (Source IP Address Register) (WIZnet, 2011, σ. 20). Η αλλαγή της τιμής κάποιου, έχει άμεση εφαρμογή στη διεπαφή, γεγονός που λαμβάνεται υπόψη σε σχετικά εισερχόμενα αιτήματα ώστε να δίνεται απόκριση με τις τρέχουσες ρυθμίσεις της διεπαφής πριν την ενημέρωσή τους.

#### Μέγεθος μνήμης ανά Socket

Το W5100 υποστηρίζει μέχρι τέσσερα, ταυτόχρονα, ενεργά Socket καθένα από τα οποία αποδίδεται ένα μέρος μνήμης από τα συνολικά διαθέσιμα 8KiB για κάθε κατεύθυνση κίνησης δεδομένων, ανεξάρτητου μεγέθους το καθένα. Οι καταχωρητές RMSR (RX Memory Size Register) και TMSR (TX Memory Size Register) καθορίζουν την προσωρινή μνήμη που αφιερώνεται σε κάθε Socket μεταξύ 1, 2, 4 ή 8 KiB. Σαφώς, πρέπει να δίνεται προσοχή ώστε ο ανατεθειμένος χώρος στα Socket να μην ξεπερνάει τα 8KiB καθώς, σε αυτήν την περίπτωση, ορισμένα Socket θα εργάζονται σε κοινές θέσεις στη μνήμη, με ότι επακόλουθα μπορεί αυτό να επιφέρει.

Το σχήμα 7.1.4 παρουσιάζει τη σημασιολογία των bit των δύο αυτών κατα-

Σχήμα 7.1.4: Καταχωρητές μεγέθους, RMSR και TMSR.

Το μέγεθος κάθε Socket καθορίζεται από την τιμή  $2^{S1:0}$ .

Socket 3	Socket 2	Socket 1	Socket 0
S1	S0	S1	S0

Σχήμα 7.1.5: Καταχωρητές κατάστασης και διακοπών, IR και IMR.

7	6	5	4	3	2	1	0
CONFLICT	UNREACH	PPPoE	-	S3_INT	S2_INT	S1_INT	S0_INT

χωρητών. Παρατηρείται ότι το μέγεθος της μνήμης κάθε Socket ρυθμίζεται από δύο, μόνο, bit. Η υλοποίηση χρησιμοποιεί μόνο ένα Socket με σκοπό τη λήψη και απόκριση αιτημάτων HTTP. Για το λόγο αυτό οι RMSR και TMSR ρυθμίζονται ώστε να αποδίδονται 8KiB τόσο για την εισερχόμενη όσο και για την εξερχόμενη προσωρινή μνήμη αποθήκευσης του Socket 0. Οι ρυθμίσεις του Socket ολοκληρώνονται στην παράγραφο Θύρα και πρωτόχολο Socket (σ. 112).

### Κατάσταση και διακοπές

Δύο τελευταίοι καταχωρητές που εξετάζονται είναι οι IR (Interrupt Register) και IMR (Interrupt Mask Register), οι οποίοι καθορίζουν την κατάσταση του W5100 και την αναγγελία διακοπών, αντιστοίχως. τα bit του καταχωρητή IR – μόνο για ανάγνωση – τίθενται σε κάθε περίπτωση που το απαιτεί, ενώ διακοπή (μέσω του ακροδέκτη  $\overline{INT}$ ) προκαλείται μόνο όταν τίθενται bit του IR των οποίων το αντίστοιχο bit του IMR έχει, επίσης, τεθεί (WIZnet, 2011, σσ. 21–22). Με αυτόν τον τρόπο, αναγγέλλονται μόνο οι διακοπές που ενδιαφέρουν. Επιπλέον, ο μικροελεγκτής εξετάζοντας τον IR μπορεί, ανά πάσα στιγμή, να αποφασίσει εάν το W5100 χρειάζεται την προσοχή του.

Τα bit των καταχωρητών IR και IMR παρουσιάζονται στο σχήμα 7.1.5. Από τα συνολικά 7 διαθέσιμα bit, μόνο το S0\_INT που ειδοποιεί για αλλαγή της κατάστασης του – μοναδικού ενεργού – Socket 0 έχει ενδιαφέρον για την υλοποίηση και για το λόγο αυτό, τίθεται σε λογικό 1 στον καταχωρητή IMR. Για λεπτομέρειες σχετικά με τη φύση της διακοπής είναι απαραίτητη η εξέταση του αφιερωμένου καταχωρητή κάθε Socket.

Ένα σημαντικό σημείο είναι ότι το σήμα του ακροδέκτη  $\overline{INT}$  τίθεται και παραμένει σε λογικό 0 μέχρι να διευθετηθούν όλα τα ενεργοποιημένα, για διακοπή, bit του IR. Το χαρακτηριστικό αυτό χρησιμοποιείται, στο πλαίσιο της υλοποίησης, σε συνδυασμό με έναν από τους ακροδέκτες INTn του μικροελεγκτή, ώστε η ρουτίνα

εξυπηρέτησης διακοπής να εκτελέσται σε σήμα λογικού 0, και όχι σε παρυφή. Η διευθέτηση αυτή έχει το πλεονέκτημα ότι ακόμα και εάν ο μικροελεγκτής έχει τεθεί σε κατάσταση χαμηλής κατανάλωσης (Power-down), είναι βέβαιο ότι το σήμα στον INT θα προκαλέσει, εκτός από αφύπνιση της CPU, και την εκτέλεση της αντίστοιχης ρουτίνας εξυπηρέτησης. Όπως αναφέρεται στο εγχειρίδιο της Atmel (2013, σ. 71), το τελευταίο είναι κάτι που πρέπει να ληφθεί υπόψη όταν χρησιμοποιούνται παρυφές για την αφύπνιση καθώς, εάν η διάρκεια τους είναι σύντομη, παρότι η CPU θα αφυπνιστεί, ενδέχεται να μην αναγνωριστεί η διακοπή. Έτσι, εξαλείφεται αυτό το ενδεχόμενο.

### Θύρα και πρωτόκολλο Socket

Ένα Socket για να λειτουργήσει χρειάζεται, εκτός από διεύθυνση IP, αριθμό θύρας. Η διεύθυνση που χρησιμοποιούν όλα τα ενεργά Socket του W5100 είναι αυτή που καθορίζεται από τους Καταχωρητές διεύθυνσεων και μάσκας υποδικτύου (σ. 110). Ωστόσο, ο αριθμός θύρας, προσδιορίζεται για κάθε Socket ξεχωριστά από τον καταχωρητή Sn\_PORT.

Επιπλέον του αριθμού θύρας, το W5100 ρυθμίζεται με το πρωτόκολλο που διεκπεραιώνει κάθε Socket. Η ρύθμιση γίνεται από τα τέσσερα τελευταία bit του καταχωρητή Sn\_MR (Socket n Mode Register) (WIZnet, 2011, σσ. 25–26). Στην περίπτωση της υλοποίησης, το μοναδικό ενεργό Socket προορίζεται για παραλαβή αιτημάτων HTTP. Σύμφωνα με τους R. Fielding κ.ά. (1999, σ. 13), το HTTP θεωρεί ότι το υποκείμενο πρωτόκολλο παρέχει αξιόπιστη μεταφορά, και είνισται να είναι το TCP, με αριθμό θύρας 80. Η υλοποίηση ακολουθεί αυτές τις καθιερωμένες πρακτικές.

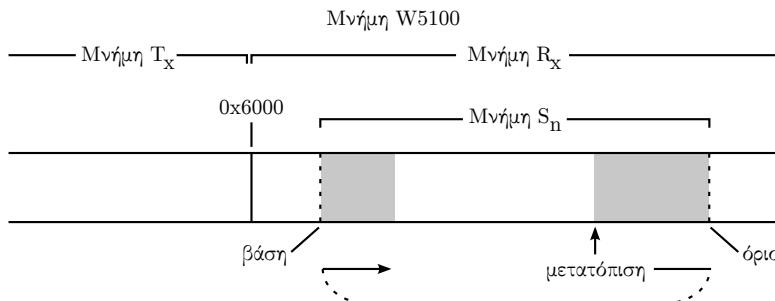
### Διαχείριση μνήμης Socket

Στην παράγραφο Μέγεθος μνήμης ανά Socket (σ. 110) περιγράφεται ο τρόπος απόδοσης μνήμης σε κάθε Socket για εισερχόμενα και εξερχόμενα δεδομένα. Η λήψη και αποστολή των Byte αναλαμβάνεται, σαφώς, από το W5100. Ωστόσο, το λογισμικό του μικροελεγκτή είναι υπεύθυνο για την ανάγνωση και εγγραφή των κατάλληλων, κάθισ φορά, διεύθυνσεων της μνήμης του W5100.

Για την περίπτωση της μνήμης εισερχομένων, τα Socket διαθέτουν τους καταχωρητές Sn\_RX\_RSR (Socket n Rx Receive Size Register) και Sn\_RX\_RR (Socket n Rx Read Register). Ο πρώτος αναγγέλλει το πλήθος, ενώ από το δεύτερο είναι δυνατό να αναχθεί η διεύθυνση του πρώτου προς ανάγνωση Byte, χωρίς, ωστόσο, να περιέχει ο ίδιος τη φυσική αυτή διεύθυνση (WIZnet, 2011, σσ. 35–36). Σε σχέση με το τελευταίο, η αφιερωμένη μνήμη κάθε Socket (Μνήμη S<sub>n</sub>, στο σχήμα 7.1.6) λειτουργεί ως δακτύλιος (κυκλική ουρά). Τα δεδομένα τοποθετού-

Σχήμα 7.1.6: Μνήμη εισερχομένων για κάποιο Socket n.

Όλα τα Socket αποδίδονται μέρος της κοινής μνήμης  $R_X$ . Η μνήμη  $S_n$  κάθε Socket συμπεριφέρεται ως διαδικτύους. Η γκρι περιοχή της εικόνας αντιπροσωπεύει δεδομένα.



νται διαδοχικά από τη βάση μέχρι το όριο (βλ. ίδιο σχήμα) και, μετά, πάλι από τη βάση. Για κάθε νέο προστιθέμενο Byte, η τιμή του  $Sn\_RX\_RR$  αυξάνει κατά μία μονάδα, ανεξαρτήτως εάν η αφιερωμένη μνήμη έχει γεμίσει ή όχι. Επομένως, η μετατόπιση (offset) του πρώτου Byte (σε σχέση με τη βάση) είναι το υπόλοιπο της διάρεσης της τιμής του  $Sn\_RX\_RR$  δια το μέγεθος του Socket.

Οστόσο, επειδή οι δυνατές τιμές για το μέγεθος των Socket είναι δυνάμεις του 2 (συγκεκριμένα, 1, 2, 4 ή 8KiB), η διάρεση καθίσταται ισοδύναμη με την ακόλουθη λογική πράξη:

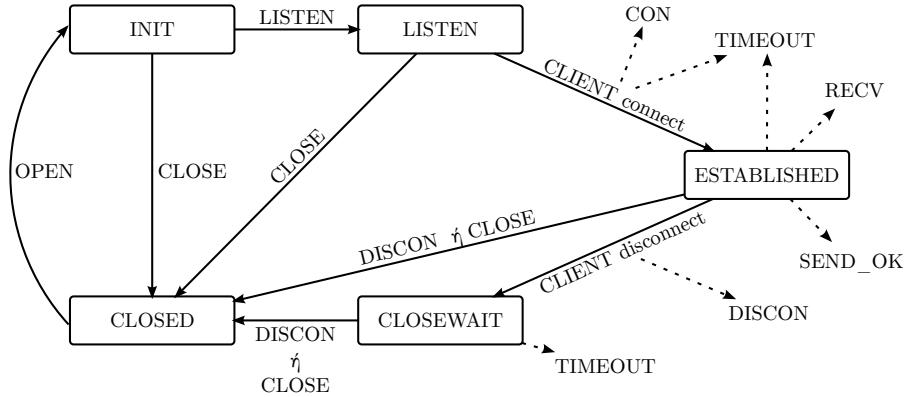
$$\text{μετατόπιση} = (Sn\_RX\_RR) \wedge (Sn\_mask)$$

όπου  $Sn\_mask$  είναι η μέγιστη αποδεκτή τιμή μετατόπισης (offset), πρακτικά, το μέγεθος μνήμης του Socket μειωμένο κατά 1. Επιπλέον, η χρήση δυνάμεων του 2 επιτρέπουν τον καταχωρητής  $Sn\_RX\_RR$  να αυξάνεται επί άπειρον χωρίς οι ενδιάμεσες υπερχειλίσεις του να προκαλούν προβλήματα. Εικάζεται ότι αυτός είναι ο τρόπος λειτουργίας του  $Sn\_RX\_RR$  με σκοπό τη μείωση της πολυπλοκότητας που εισάγει η δυνατότητα μεταβλητού μεγέθους μνήμης ανά Socket. Στο εγχειρίδιο της WIZnet (2011, σσ. 35,43–44) αναφέρεται η ύπαρξη και εφαρμογή της μάσκας χωρίς να αιτιολογείται ο λόγος.

Πλέον, το άθροισμα μεταξύ μετατόπισης και βάσης (δηλαδή, της φυσική διεύθυνσης της πρώτης θέσης της μνήμης  $S_n$ ), δίνουν τη φυσική διεύθυνση του πρώτου Byte προς ανάγνωση. Προφανώς, η τιμή της βάσης εξαρτάται από τα πόσα Byte έχουν αποδοθεί σε κάθε προηγούμενο Socket του τρέχοντος.

Σημειώνεται το προφανές, ότι το λογισμικό του ελεγκτή είναι υπεύθυνο να αναγνωρίζει εάν τα διαθέσιμα Byte βρίσκονται όλα σε διαδοχικές θέσεις μνήμης ή εάν χωρίζονται σε δύο μέρη, όπως στην περίπτωση που απεικονίζεται στο σχήμα

Σχήμα 7.1.7: Γράφος καταστάσεων Socket διαχομιστή HTTP.



Βασισμένο. Main Sn\_SR statuses. Στο WIZnet. W5100 Datasheet. Έκδ. 1.2.4. 20 Σεπτ. 2011. 71 σσ. url: [http://www.wiznet.co.kr/Sub\\_Modules/en/product/product\\_detail.asp?Refid=653&page=1&cate1=5&cate2=7&cate3=26&pid=1011&cType=2](http://www.wiznet.co.kr/Sub_Modules/en/product/product_detail.asp?Refid=653&page=1&cate1=5&cate2=7&cate3=26&pid=1011&cType=2) (επίσημη 25/01/2014), σ. 28

7.1.6. Σε κάθε περίπτωση, ο μικροελεγκτής αυξάνει την τιμή του Sn\_RX\_RR σύμφωνα με το πλήθος των Byte που έχει αναγνώσει και αποστέλλει την εντολή RECV ώστε να ενημερωθεί το W5100 για το χώρο που είναι ξανά διαθέσιμος για νέα δεδομένα.

Η μνήμη εξερχομένων λειτουργεί με τον ίδιο, κατά βάση, τρόπο που περιγράφεται παραπάνω. Οι αντίστοιχοι καταχωρητές είναι οι Sn\_TX\_FSR (Socket n Tx Free Size Register) και Sn\_TX\_WR (Socket n Tx Write Register), ενώ η εντολή που ειδοποιεί το W5100 για την ανάγκη αποστολής των εγγεγραμμένων Byte στη μνήμη T<sub>x</sub> είναι η SEND.

### Χειρισμός Socket

Προκειμένου το Socket να πραγματοποιήσει κάποια ενέργεια (για παράδειγμα, αποστολή δεδομένων), ο κωδικός της ενέργειας αυτής πρέπει να εγγραφεί στον αντίστοιχο καταχωρητή Sn\_CR (Socket n Command Register), ενώ η κατάσταση του Socket την κάθε στιγμή, αποφαίνεται από την τιμή του καταχωρητή Sn\_SR (Socket n Status Register) (WIZnet, 2011, σσ. 26–30). Επιπροσθέτως, ο καταχωρητής Sn\_IR (Socket n Interrupt Register) διαθέτει μία σειρά ενδείξεων που πληροφορούν την αιτία αναγγελίας διακοπής μέσω του ακροδέκτη INT. όσο έστω και μία από αυτές τις ενδείξεις βρίσκεται σε λογικό 1, το bit Sn\_INT του καταχωρητή IR παραμένει ενεργό, ενώ ο μηδενισμός κάποιας γίνεται γράφοντας λογικό 1 στο επιισυμητό bit (WIZnet, 2011, σ. 27).

Στο σχήμα 7.1.7 εμφανίζονται οι βασικές καταστάσεις (πλαίσια), οι εντολές (συμπαγή βέλη) και τα αίτια πρόκλησης διακοπής (διάστικτα βέλη) για το Socket

του διακομιστή HTTP. Σημειώνεται ότι τα CLIENT connect και CLIENT disconnect αποστέλλονται από τον πελάτη και αποτελούν αιτήματα έναρξης και λήξης σύνδεσης TCP.

Αρχικά, εφόσον έχει αρχικοποιηθεί το Socket, στέλνοντας την εντολή LISTEN, το Socket τίθεται σε κατάσταση αναμονής αιτημάτων και παραμένει σε αυτή έως ότου καταφύγει αίτημα σύνδεσης. Κατά τη διάρκεια εγκαθίδρυσης της σύνδεσης, το W5100 θέτει το bit CON του καταχωρητή Sn\_IR προκαλλώντας διακοπή (δεδομένου ότι έχουν ενεργοποιηθεί οι διακοπές για αυτό το Socket) (βλ. Κατάσταση και διακοπές, σ. 111).

Σε κατάσταση ενεργούς σύνδεσης, ο πελάτης στέλνει πακέτα, το φορτίο των οποίων τοποθετείται στην αφειρωμένη μνήμη εισερχομένων και ενεργοποιείται το bit RCV του Sn\_IR. Σύμφωνα με το εγχειρίδιο της WIZnet (2011, σ. 28), το RCV παραμένει ενεργό όσο υπάρχουν διαθέσιμα δεδομένα για ανάγνωση. Τυπικά, η επεξεργασία των εισερχόμενων δεδομένων προκαλεί την παραγωγή δεδομένων απόκρισης που τοποθετούνται στην μνήμη εξερχομένων και αποστέλλονται υποβάλλοντας την εντολή SEND (Sn\_CR). Το bit SEND\_OK (Sn\_IR) ειδοποιεί ότι όλα τα δεδομένα της μνήμης εξερχομένων έχουν σταλεί. Η διαδικασία ανταλλαγής συνεχίζει για όσο απαιτείται.

Τελικά, η σύνδεση τερματίζεται είτε ως αποτέλεσμα αιτήματος του πελάτη είτε με την εντολή DISCON. Κατόπιν, το Socket αρχικοποιείται και τίθεται, εκ νέου, σε κατάσταση αναμονής (passive open).

Στο σχήμα αναφέρεται ένα ακόμα αίτιο διακοπής· η λήξη του χρόνου αναμονής (TIMEOUT) κατά την αρχική ή τελική τριπλή χειραψία ή κατά την ανταλλαγή δεδομένων. Σε κάθε περίπτωση, η αντιμετώπιση του συμβάντος, από πλευράς της υλοποίησης, είναι, πάντα, ο τερματισμός της σύνδεσης και το κλείσιμο του Socket ακολουθούμενο από την μετέπειτα επανενεργοποίησή του.

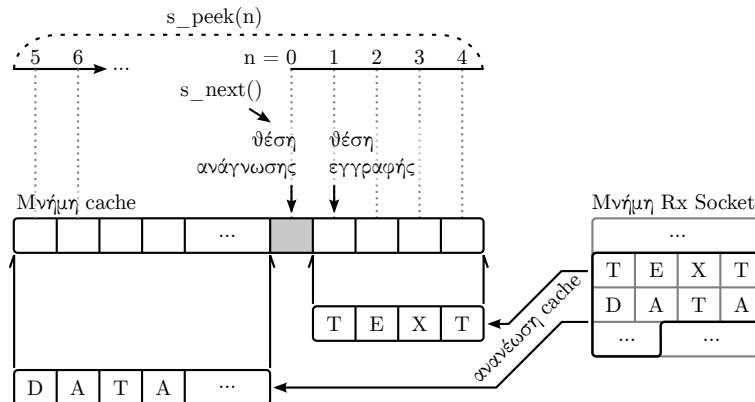
Τυπικά, η υλοποίηση ενδιαφέρεται μόνο για διακοπές που οφείλονται σε εισερχόμενα δεδομένα, τερματισμό σύνδεσης ή στο πέρας του χρόνου αναμονής. Πάντως, ανεξαρτήτως περίπτωσης, η ρουτίνα εξυπηρέτησης διακοπών του ακροδέκτη INT, μεριμνεί για τον καθαρισμό των ενδείξεων που την προκάλεσαν ώστε να απενεργοποιείται το σήμα του ακροδέκτη μέχρι το επόμενο συμβάν.

#### 7.1.4 Συμπληρωματικό λογισμικό W5100

Το λογισμικό για την ανταλλαγή δεδομένων μεταξύ μικροελεγκτή και W5100 εργάζεται με προσωρινούς χώρους αποθήκευσης· μία διεύθυνση στην κύρια μνήμη του μικροελεγκτή και το πλήθος των Byte που εκτείνονται ξεκινώντας από αυτήν, είναι αρκετά για την παραλαβή ή αποστολή μερικών Byte. Με διαδοχικές κλήσεις είναι δυνατό να παραληφθεί ή να αποσταλεί το συνολικό φορτίο. Ωστόσο, στην

Σχήμα 7.1.8: Ροή χαρακτήρων από τη μνήμη εισερχομένων Socket.

Οι `s_peek()` και `s_next()` αναπαριστούν την ανάγνωση ενός οποιουδήποτε χαρακτήρα και την εξαγωγή του επόμενου, αντιστοίχως.



περίπτωση του HTTP Socket, τα εισερχόμενα δεδομένα προορίζονται για λεξική ανάλυση, στοιχειώδους έστω μορφής, ώστε να αναγνωρίζεται η εννοιολογική διάσταση της κάθε αναπαράστασης.

### Ροή χαρακτήρων

Σε αυτήν την περίπτωση, η εργασία με τον οποιοδήποτε προσωρινό χώρο αποθήκευσης είναι προτιμότερο να αποκρύπτεται από τους αναλυτές, παρέχοντας μία διεπαφή για τη σειριακή εξαγωγή Byte ή, στην προκειμένη, χαρακτήρων, τον ένα μετά τον άλλο (ροή χαρακτήρων – character stream). Εναλλακτικά, και χωρίς τήρηση προσωρινού χώρου αποθήκευσης, η διεπαφή μπορεί να κάνει άμεση χρήση της υποδομής για την ανταλλαγή δεδομένων με το W5100. Ωστόσο, η ταχτική αυτή επιβαρύνει την εξαγωγή κάθε χαρακτήρα με τους υπολογισμούς που αναφέρονται στην Διαχείριση μνήμης Socket (σ. 112). Αντιθέτως, προτιμάται η τήρηση τμημάτων του εισερχόμενου φορτίου σε προσωρινό χώρο, cache, τοπικά στη μνήμη του μικροελεγκτή ώστε η οποιαδήποτε επιβάρυνση να υφίσταται μόνο κατά την ενημέρωση της cache με την επόμενη ομάδα χαρακτήρων.

Επιπροσθέτως, για την υποστήριξη συμπληρωματικών δυνατοτήτων για την περαιτέρω διευκόλυνση των αναλυτών, όπως η ανάγνωση κάποιου επόμενου χαρακτήρα (look forward) (δηλαδή, χωρίς την υποχρεωτική εξαγωγή όλων των ενδιάμεσων), χωρίς την υποβάθμιση των επιδόσεών της, η προσωρινή μνήμη υλοποιείται ως δακτύλιος (κυκλική ουρά). Τα πλεονεκτήματα χρήσης δακτυλίου γίνονται περισσότερο αντιληπτά λαμβάνοντας υπόψη το σχήμα 7.1.8.

Οι χαρακτήρες τοποθετούνται στη μνήμη cache σε διαδοχικές θέσεις και εκτεί-

νονται από τη θέση ανάγνωσης προς το τέλος του αφιερωμένου χώρου μέχρι (αλλά μη συμπεριλαμβανομένης) τη θέση εγγραφής. Κάθε φορά που εξάγεται ένας χαρακτήρας από την cache, η θέση ανάγνωσης αυξάνεται κατά μία θέση ώστε να προσδιορίζει τον επόμενο για εξαγωγή ή ανάγνωση χαρακτήρα. Αντιστοίχως, όποτε διαβάζεται μία ομάδα χαρακτήρων από τη μνήμη του Socket, η θέση εγγραφής αυξάνεται κατά πλήθος ίσο με αυτό που ελήφθη, ώστε να καταδεικνύεται η επόμενη της τελευταίας θέσης που ενημερώθηκε με νέο χαρακτήρα. Και οι δύο ενδείξεις θέσης εκτελούν κυκλική κίνηση, με την έννοια ότι η επόμενη θέση της τελευταίας διαθέσιμης του αφιερωμένου χώρου είναι πάντα η πρώτη (θέση 0).

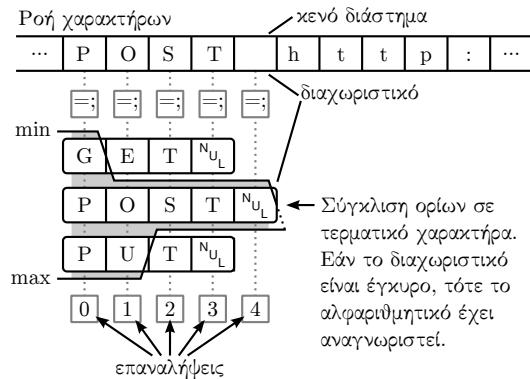
Προφανώς, στην περίπτωση κατά την οποία η θέση ανάγνωσης αυξηθεί και ταυτόχρονα εξισωθεί με τη θέση εγγραφής, η μνήμη cache έχει εξαντληθεί, με αποτέλεσμα στην επόμενη χρήση της διεπαφής, να γίνεται προσπάθεια ανάκτησης της επόμενης ομάδας χαρακτήρων. Το πλήθος των λαμβανόμενων χαρακτήρων εξαρτάται, σαφώς, από τους εναπομείναντες χαρακτήρες στη μνήμη του Socket και του μη δεσμευμένου χώρου στην cache (ο μικρότερος εκ των δύο). Εάν η εξαντληση της cache συμβαίνει πάντα στην τελευταία φυσική θέση του αφιερωμένου χώρου, η χρήση δακτυλίου είναι περιττή.

Οστόσο, ενδέχεται να προκύψει ανάγκη ανανέωσης πριν την εξαντληση των αποθεμάτων της cache. Τέτοιες περιπτώσεις οφείλονται στη χρήση της δυνατότητας ανάγνωσης (δίχως εξαγωγή), όταν η επιλυμητή θέση βρίσκεται έξω από το (ενημερωμένο) εύρος της cache. Όπως στο παράδειγμα του σχήματος 7.1.8, η ανάγνωση οποιουδήποτε χαρακτήρα έξω από το ενημερωμένο εύρος (δηλαδή, για  $n \geq 1$ ) προϋποθέτει την ανανέωση της cache. Μετά την ολοκλήρωση της πρόωρης ανανέωσης, η θέση εγγραφής καταλήγει, τις περισσότερες φορές, σε θέση διαφορετική της θέσης 0. Αυτό ακριβώς το χαρακτηριστικό – τα ολισθηρά άκρα – του δακτυλίου εγγυάται τη συνεχόμενη εισαγωγή και εξαγωγή χαρακτήρων.

Εάν η μνήμη cache υλοποιηθεί ως (απλή) ουρά, η ανάγκη για πρόωρη ενημέρωση ενέχει την ενδεχόμενη μετατόπιση των υπαρχόντων χαρακτήρων στις πρώτες θέσεις του πίνακα ώστε να δημιουργηθεί χώρος για τους νέους χαρακτήρες στο τέλος του πίνακα. Ωστόσο, κρίνεται ότι αυτή η τακτική είναι λιγότερο αποδοτική και για αυτόν το λόγο προτιμάται η χρήση δακτυλίου.

Σε κάθε περίπτωση υλοποίησης, η μέγιστη υποστηριζόμενη θέση ανάγνωσης ( $n$ ) περιορίζεται από το συνολικό μέγεθος της cache το οποίο καθορίζεται κατά τη σύνταξη του προγράμματος. Επίσης, στην περίπτωση του δακτυλίου, απαιτούνται το πολύ δύο αναγνώσεις της μνήμης του Socket για την ενημέρωση της cache. Τέλος, σε περίπτωση που τα δεδομένα της μνήμης του Socket έχει εξαντληθεί, όλες οι συναρτήσεις της διεπαφής επιστρέφουν μία σχετική ένδειξη (EOF).

Σχήμα 7.1.9: Λεξική ανάλυση ροής Socket.



### Λεξική ανάλυση ροής

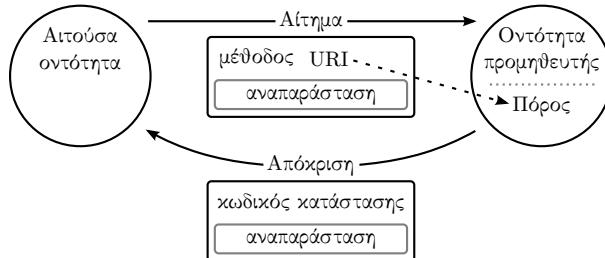
Μία αναμενόμενη λειτουργία των αναλυτών είναι η επεξεργασία των εισερχόμενων χαρακτήρων της ροής για την αναγωγή της παρεχόμενης πληροφορίας σε μορφή εύκολα εκμεταλλεύσιμης από το λογισμικό.

Ο μηχανισμός της υλοποίησης που παρέχεται για αυτόν τον σκοπό βασίζεται στην παραδοχή ότι κάθε στιγμή που προκύπτει ανάγκη για την αναγνώριση των εισερχομένων, υπάρχει εκτίμηση σχετικά με το τι αναμένεται να παραληφθεί (δηλαδή, ποιες είναι οι πιθανές αποδεκτές είσοδοι). Συνεπώς, ένα μέρος της ροής χαρακτήρων του Socket αντιπαραβάλλεται με ένα προκαθορισμένο σύνολο πιθανών αλφαριθμητικών. Η επιλογή των «πιθανών» αυτών αλφαριθμητικών γίνεται με γνώμονα του τι αρμόζει στην κάθε περίπτωση ενεργοποίησης του μηχανισμού.

Χαρακτηριστική περίπτωση χρήσης του μηχανισμού είναι η ανάλυση ενός αιτήματος HTTP. Κάθε αίτημα HTTP αρχίζει με τη γραμμή αιτήματος η οποία, το πρώτο πράγμα που προσδιορίζει είναι η μέθοδος για την επενέργεια στο ακολουθούμενο URI (R. Fielding κ.ά., 1999, σ. 35). Το HTTP/1.1 ορίζει οκτώ, συνολικά, μεθόδους, εκ των οποίων ο διακομιστής μπορεί να τις υποστηρίζει όλες ή ένα υποσύνολό τους. Όπως είναι αναμενόμενο, κάθε υποστηριζόμενη και, συνεπώς, αναγνωρίσιμη μέθοδος πρέπει να ορίζεται ως αλφαριθμητικό, ώστε να είναι δυνατή η αντιπαραβολή της με τους εισερχόμενους χαρακτήρες. Παρότι τα αλφαριθμητικά αποθηκεύονται ανεξάρτητα το ένα από το άλλο, διατηρείται, ωστόσο, μία συλλογή με αναφορές προς αυτά. Για την απλοποίηση του υποκείμενου μηχανισμού, οι αναφορές τοποθετούνται έτσι ώστε τα αλφαριθμητικά να εμφανίζονται σε αλφαριθμητική σειρά και το καθένα εμφανίζεται μόνο μία φορά.

Όταν, επομένως, καταφύγει ένα νέο αίτημα, είναι βέβαιο ότι το πρώτο πράγμα που πρέπει να προσδιοριστεί είναι η μέθοδος HTTP, με αποτέλεσμα ο μηχανισμός

Σχήμα 7.2.1: Αίτημα και απόχριση μεταξύ αιτούσας και οντότητας προμηθευτή.



να ενεργοποιείται με τη συλλογή αλφαριθμητικών των αποδεκτών μεθόδων. Αρχικά, συμμετέχει ολόκληρο το σύνολο, όπως καθορίζεται από τα όρια `min` και `max` (σχήμα 7.1.9).

Η αντιπαροβολή των χαρακτήρων πραγματοποιείται σε επαναλήψεις όπου σε καθεμία, εξάγεται ένας χαρακτήρας από τη ροή και συγχρίνεται με τον επόμενο χαρακτήρα των διαθέσιμων αλφαριθμητικών (δηλαδή, των αλφαριθμητικών μεταξύ `min` και `max`). Επειδή τα αλφαριθμητικά παρέχονται σε αλφαριθμητική σειρά, τα όρια είναι δυνατό να ολισθαίνουν εξαιρώντας όσα αποτυγχάνουν κάποια σύγκριση.

Με τον τρόπο αυτό, τα όρια, σταδιακά, συγχλίνουν προς μία θέση, καταδεικνύοντας το αλφαριθμητικό που σημειώνει μόνο επιτυχούσες συγκρίσεις. Εφόσον οι επαναλήψεις τερματίστοιν εξαιτίας της εξάντλησης των χαρακτήρων του εν λόγω αλφαριθμητικού, ενώ έχει εντοπιστεί στη ροή έγκυρος διαχωριστικός χαρακτήρας (delimiter) (στην προκειμένη, το κενό διάστημα), η μέθοδος έχει αναγνωριστεί επιτυχώς. Ένα σημαντικό σημείο είναι ότι, εφόσον ολοκληρωθεί επιτυχώς η λεξική αναγνώριση, ο εντοπισμένος όρος περιγράφεται αυτομάτως από τη θέση που κατέχει στην τρέχουσα συλλογή, κάτι που διευκολύνει την αναγγελία του σε λοιπά μέρη της υλοποίησης, καθώς χρησιμοποιείται κωδικός αριθμός αντί για αλφαριθμητικό.

Τελικά, η ροή δύναται να υποβληθεί εκ νέου στην ίδια διαδικασία, αυτή τη φορά με νέα συλλογή αλφαριθμητικών, για παράδειγμα, για την αναγνώριση του αιτούμενου πόρου.

## 7.2 Υποσύστημα διακομιστή HTTP

Ο Παγκόσμιος Ιστός στηρίζεται στην ανταλλαγή μηνυμάτων μεταξύ μίας αιτούσας οντότητας (Requester entity) και μίας οντότητας προμηθευτή (Provider entity) με σκοπό την επενέργεια της πρώτης σε στοιχεία, ή γενικά, πόρους, της δεύτερης (Booth κ.ά., 2004).

Τα αποστελλόμενα μηνύματα της πρώτης, τα αιτήματα, φέρουν, κατά ελάχιστον, την ενέργεια και ένα αναγνωριστικό του πόρου στον οποίο επιδιώκεται να εφαρμοστεί. Ο προμηθευτής επεξεργάζεται το αίτημα και επιστρέφει ένα μήνυμα απόκρισης. Και στις δύο περιπτώσεις, τα μηνύματα ενδέχεται να φέρουν μία αναπαράσταση του πόρου που περιγράφει την επιδιωκόμενη ή την τρέχουσα κατάστασή του, την τιμή κάποιου άλλου πόρου ή κάποιου σφάλματος (R. T. Fielding, 2000, σσ. 90–92).

Οι κανόνες που διέπουν τις λεπτομέρειες της ανταλλαγής καθορίζονται από το πρωτόκολλο HTTP (Hypertext Transfer Protocol). Στο πλαίσιό του, ορίζονται οι πιθανές ενέργειες, ή μέθοδοι, που μπορούν να χρησιμοποιηθούν στα αιτήματα και η σημασία τους καθώς και ότι τα αναγνωριστικά έχουν τη μορφή URI (R. Fielding κ.ά., 1999, σσ. 36–37, 52–57). Το σχήμα 7.2.1 παρουσιάζει την αλληλεπίδραση μεταξύ αιτούσας και οντότητας προμηθευτή όπως περιγράφεται πιο πάνω.

Στο πλαίσιο της υλοποίησης, το υποσύστημα διακομιστή HTTP αναλαμβάνει τις αρμοδιότητες της οντότητας προμηθευτή. Πρόκειται για ένα σύμπλεγμα μονάδων όπου η καθεμία αναλαμβάνει την περάτωση ενός μέρους της συνολικής εργασίας. Το σχήμα 7.2.2 αποτυπώνει τη γενική δομή και διασύνδεση των μονάδων, ενώ οι λεπτομέρειες της λειτουργίας τους αναλύονται στη συνέχεια. Σημειώνεται ότι, παρότι στο σχήμα παρουσιάζεται ότι συμμετέχουν όλες οι μονάδες για την διεκπεραίωση ενός αιτήματος, ωστόσο, στην πραγματικότητα, η μονάδα αναπαράστασης (Representation Module) ενδέχεται να παραληφθεί. Η συμπερίληψή της ή μη καθορίζεται από τις ανάγκες του αιτήματος όπως εάν αναμένεται αναπαράσταση πόρου στο αίτημα ή εάν πρόκειται να επιστραφεί μία στο σώμα της απόκρισης.

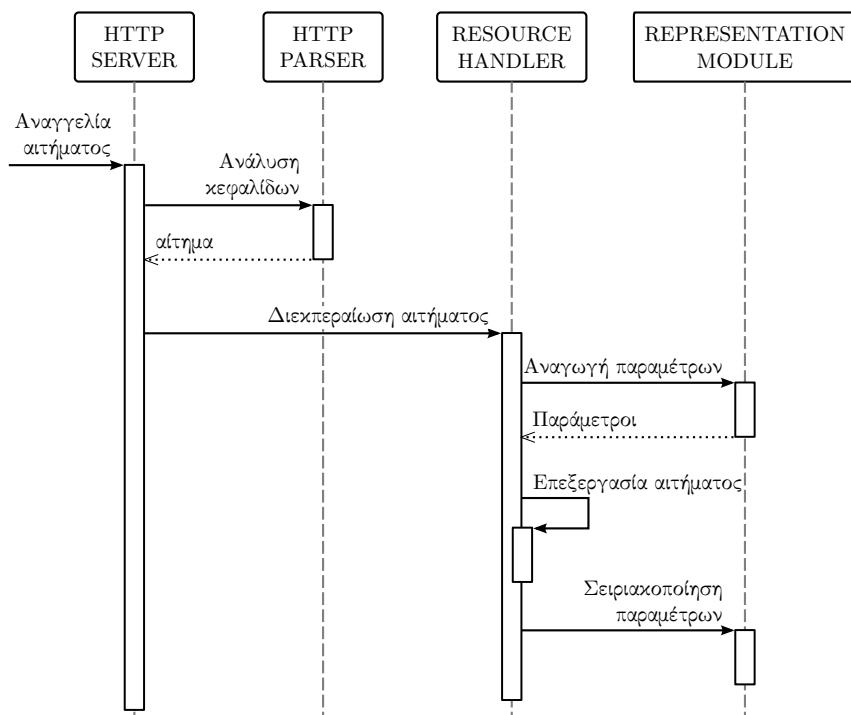
### 7.2.1 Κορμός διακομιστή

Εφόσον αναγνωριστεί ύπαρξη εισερχόμενων δεδομένων στο Socket του διακομιστή HTTP, το συμβάν αναγγέλλεται για την εκκίνηση των εργασιών διεκπεραίωσης του. Από τη βάση του διακομιστή πυροδοτείται ο αναλυτής HTTP για την αναγωγή της δομής `HTTPRequest` σύμφωνα με την κεφαλίδα του αιτήματος. Η δομή διατηρεί τα βασικά στοιχεία που ενδιαφέρουν το διακομιστή και τις ρουτίνες περάτωσης για την διεκπεραίωση του αιτήματος.

#### Γενική συμπεριφορά διακομιστή

Στην πορεία ελέγχεται η πληρότητα του αιτήματος στο σύνολό του και, σε περίπτωση κάποιου μη αποδεκτού στοιχείου, επιστρέφονται γενικής φύσεως διαγνωστικά μηνύματα σφάλματος, όπως στην περίπτωση μη αναγνωρίσιμου URI ή μη διαλέσιμης μενόδου. Οι κωδικοί κατάστασης HTTP, αντιστοίχως, είναι 404 (Not

Σχήμα 7.2.2: Διασύνδεση μεταξύ βασικών μονάδων του υποσυστήματος διαχομιστή HTTP.



Found) και 405 (Method Not Allowed), ενώ στην τελευταία περίπτωση συμπεριλαμβάνεται στην απόκριση το πεδίο κεφαλίδας «Allow» δηλώνοντας τις διαθέσιμες μεθόδους για το εκάστοτε URI, όπως προτρέπουν οι R. Fielding κ.ά. (1999, σ. 66).

Εφόσον τα γενικά χαρακτηριστικά του αιτήματος είναι αποδεκτά, η δομή πρωθείται στην κατάλληλη ρουτίνα περάτωσης, όπως δηλώνεται κατά τη ρύθμιση του διαχομιστή. Οι ρουτίνες περάτωσης είναι ελεύθερες και υποχρεωμένες να παράξουν μία αρμόζουσα απόκριση HTTP βάσει της παρεχόμενης δομής `HTTPRequest`. Με την ολοκλήρωση της εκτέλεσής τους, ολοκληρώνεται και ο κύκλος του διαχομιστή.

### Τυποδομή σύνταξης κεφαλίδας

Ένα κοινό χαρακτηριστικό όλων των παραγόμενων αποκρίσεων είναι η ύπαρξη της κεφαλίδας HTTP. Παρότι τα μηνύματα είναι διαφορετικά σε χάθε περίπτωση, ωστόσο όλες οι παραγόμενες κεφαλίδες των αποκρίσεων HTTP αποτελούνται, σε μεγάλο βαθμό, από κοινά πεδία, ενδεχομένως με διαφορετική τιμή. Προκειμένου να εξοικονομείται χώρος μνήμης που διαφορετικά θα δεσμευόταν για κάθε εμφάνιση πανομοιότυπων αλφαριθμητικών εντός διαφορετικών μονάδων μεταγλώττισης, παρέχεται μία κεντρική υποδομή για τη σύνταξη των κεφαλίδων.

Η βασική παραδοχή είναι ότι ένα πλήθος συχνά χρησιμοποιούμενων δομικών αλφαριθμητικών μονάδων διατηρείται σε μία κεντρική δεξαμενή και παρέχεται ένας κωδικός αναγνώρισης για την καθεμία. Για τη σύνθεση της επιμυητής ακολουθίας, είναι αρκετή η χρήση της διεπαφής παρέχοντας τους αντίστοιχους κωδικούς σε κατάλληλη σειρά. Εκτός από προκαθορισμένα αλφαριθμητικά, είναι δυνατή η χρήση ειδικών κωδικών που επιτρέπουν τη συμπερίληψη αυθαίρετων αλφαριθμητικών καθώς και αριθμών που μετατρέπονται σε κείμενο πριν την αποστολή τους στο Socket (για την ανταλλαγή δεδομένων Socket, βλ. Διαχείριση μνήμης Socket σ. 112).

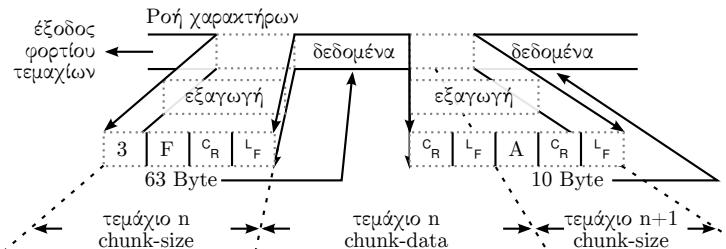
Για περισσότερη ευχρηστία, η παρεχόμενη διεπαφή πρέπει να είναι ευέλικτη σε σχέση με το πλήθος των αποδεχόμενων κωδικών. Ο τρόπος με τον οποίο επιτυγχάνεται είναι ο ορισμός και χρήση συναρτήσεων μεταβλητού πλήθους ορισμάτων (varargs ή variadic) (ISO/IEC, 2007).

### Ανασύνθεση τεμαχισμένων μηνυμάτων

Το HTTP/1.1 ορίζει μηχανισμό για τη διάσπαση του σώματος ενός μηνύματος σε τεμάχια καθένα από τα οποία συνοδεύεται από το δικό του αναγνωριστικό μεγέθους, και επιβάλλεται, στο πλαίσιο του HTTP/1.1, όλες οι υλοποίησεις να υποστηρίζουν την αποδοχή τέτοιων μηνυμάτων (R. Fielding κ.ά., 1999, σσ. 25–26). Οι βασικοί κανόνες της δομή του μηνύματος σε αυτήν την περίπτωση, παρουσιάζονται παρακάτω.

Σχήμα 7.2.3: Μηχανισμός ανασύνθεσης τεμαχισμένου σώματος μηνύματος HTTP.

Ο μηχανισμός στηρίζεται και παρέχει συμβατή (ίδια) διεπαφή με το μηχανισμό για την εξαγωγή χαρακτήρων από τη ροή (Ροή χαρακτήρων σ. 116).



```

Chunked-Body = *chunk
               last-chunk
               trailer
               CRLF
chunk = chunk-size [ chunk-extension ] CRLF
       chunk-data CRLF
chunk-size = 1*HEX
last-chunk = 1*(`0') [ chunk-extension ] CRLF
chunk-data = chunk-size(OCTET)
trailer = *(entity-header CRLF)

```

Η ύπαρξη τεμαχισμένου σώματος δηλώνεται από το πεδίο κεφαλίδας «Transfer-Encoding» με τιμή «chunked». Σε περίπτωση λήψης ενός μηνύματος με σχετική ένδειξη, πριν την αλήση της αρμόδιας ρουτίνας περάτωσης, οι αναλυτές διευθετούνται ώστε να χρησιμοποιούν έναν ελαφρώς διαφορετικό μηχανισμό για την εξαγωγή χαρακτήρων από τη ροή του Socket, ώστε το σώμα να ανασυντάσσεται στην αρχική του μορφή χωρίς οι αναλυτές να επιβαρύνονται με τις λεπτομέρειες της διαδικασίας.

Όπως παρουσιάζεται στο σχήμα 7.2.3, ο μηχανισμός ανασύνθεσης (`c_next()`) βασίζεται στην εξαγωγή χαρακτήρων από τη ροή Socket (Ροή χαρακτήρων σ. 116) κυρίως για την αναγνώριση του μεγέθους κάθε επόμενου τεμαχίου και για την παράλειψη αυτών των χαρακτήρων από την κανονική ροή. Διαδοχικές χρήσεις της διεπαφής του μηχανισμού ανασύνθεσης επιστρέφουν το επόμενο Byte, όπως θα συνέβαινε από την απευθείας χρήση του βασικού μηχανισμού, με τη διαφορά ότι τα εξαγόμενα Byte καταμετρούνται.

Εφόσον τα Byte ενός τεμαχίου εξαντληθούν (δηλαδή, έχουν αναγνωσθεί τόσα Byte από τη ροή όσα δηλώθηκαν στο τμήμα `chunk-size`), ο μηχανισμός καταναλώνει τα Byte που αντιστοιχούν στο μέγεθος του επόμενου τεμαχίου και ο κύκλος επαναλαμβάνεται. Όταν παραληφθεί το τελευταίο τεμάχιο (μεγέθους 0),

ο μηχανισμός απορρίπτει ενδεχόμενα τερματικά πεδία (trailer), καθώς, σύμφωνα με τους R. Fielding κ.ά. (1999, σ. 26), ο αποστολέας αποδέχεται ότι τα πεδία του trailer είναι προαιρετικά και μη αναγκαία για την ορθή ερμηνεία του μηνύματος, ενώ στο πλαίσιο της υλοποίησης χρίνεται μάλλον απίθανη η παραλαβή τεμαχισμένου μηνύματος, πόσο μάλλον τερματικών πεδίων.

### Παραγωγή τεμαχισμένων μηνυμάτων

Η υποστήριξη εισερχόμενων τεμαχισμένων μηνυμάτων γίνεται, κυρίως, για λόγους συμβατότητας με τις προδιαγραφές του πρωτοκόλλου HTTP/1.1. Ωστόσο, η αντίστροφη διαδικασία – η παραγωγή αποχρίσεων με τεμαχισμένο σώμα – χρίνεται ζωτικής σημασίας ιδίως για την επιστροφή καταχωρημένων μετρήσεων. Ο διακομιστής παρέχει έναν απλό μηχανισμό στις ρουτίνες περάτωσης των αιτημάτων για τη διατύπωση του μεγέθους κάθε τεμαχίου που πρόκειται να στείλουν. Με τον τρόπο αυτό, οι ρουτίνες είναι υπεύθυνες μόνο για τη σωστή διάσπαση του σώματος της απόχρισης σε τμήματα που αυτές μπορούν να χειριστούν.

Όπως έχει αναφερθεί, η υποστήριξη τεμαχισμένων μηνυμάτων είναι υποχρεωτική για όλες τις υλοποίησεις HTTP/1.1 όχι, ωστόσο, για αρκετές παλαιότερες υλοποίησεις του HTTP/1.0 όπου το πεδίο Transfer-Encoding είναι άγνωστο (R. Fielding κ.ά., 1999, σσ. 25–26,144). Στο πλαίσιο της υλοποίησης, η αδυναμία αποστολής τεμαχισμένου σώματος συνεπάγεται αδυναμία αποστολής της αναμενόμενης απόχρισης των αντίστοιχων ρουτινών περάτωσης που κάνουν χρήση του πεδίου.

### 7.2.2 Αναλυτής HTTP

Ο αναλυτής HTTP ενεργοποιείται από το διακομιστή για την αναγωγή δομής HTTPRequest από τη ροή του Socket. Όπως ορίζουν οι R. Fielding κ.ά. (1999, σ. 35), η μορφή της κεφαλίδας ενός αιτήματος HTTP καθορίζεται από τους ακόλουθους κανόνες:

```
Request = Request-Line
         *(( general-header
           | request-header
           | entity-header ) CRLF)
         CRLF
         [ message-body ]
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Η γραμμή αιτήματος περιέχει την μέθοδο HTTP ακολουθούμενη από τον πόρο στον οποίο είναι επιθυμητό να εφαρμοστεί (για περισσότερα, βλ. Εσωτερική περιγραφή πόρων, σ. 126). Κατόπιν, ακολουθεί η έκδοση του πρωτοκόλλου HTTP

που διέπει την επικοινωνία. Η ακολουθία CRLF (χαρακτήρες Carriage Return και Line Feed), χρησιμοποιείται για να δηλώσει το τέλος οποιουδήποτε πεδίου ενώ, στην προκειμένη, το τέλος της γραμμής αιτήματος.

Μετά τη γραμμή αιτήματος, εκτείνονται τα πεδία της κεφαλίδας τα οποία παρέχουν επιπρόσθετες πληροφορίες σχετικά με το αίτημα. Κάθε γραμμή πεδίου αρχίζει με το όνομά του, ακολουθεί άνω κάτω τελεία και η τιμή του ενώ είναι δυνατό να εκτείνεται σε περισσότερες από μία γραμμές, εφόσον, κάθε γραμμή τερματίζεται από ακολουθία CRLF αμέσως ακολουθούμενη από γραμμικό κενό διάστημα (linear white space), δηλαδή, κενό διάστημα ή χαρακτήρα οριζόντιας οριοθέτησης (horizontal tab) (R. Fielding κ.ά., 1999, σσ. 31–32).

Επιπροσθέτως, ορισμένα πεδία ορίζονται ως λίστα και είναι αποδεκτό να αποδίδονται πολλαπλές τιμές όπως, για παράδειγμα, το πεδίο Accept (R. Fielding κ.ά., 1999, σσ. 32,100). Σε αυτήν την περίπτωση, οι τιμές δίνονται στο ίδιο πεδίο, διαχωρίζοντάς τα με κόμμα ή και σε επανεμφάνιση του ίδιου πεδίου (R. Fielding κ.ά., 1999, σ. 32).

Ο αναλυτής HTTP σέβεται πλήρως αυτές τις προδιαγραφές και προσπαθεί επιπρόσθετα να είναι ελαστικός σε ορισμένες πιθανές αποκλίσεις. Για παράδειγμα, αδιαφορεί για πλήθος ασήμαντων κενών ή για το εάν η μέθοδος είναι γραμμένη με κεφαλαία. Οι επιλογές αυτές γίνονται κατόπιν προτροπής των R. Fielding κ.ά. (1999, σ. 166) για περισσότερο ανεκτικές υλοποιήσεις.

Σε σχέση με το Request-URI, όπως αναφέρεται και στην Εσωτερική περιγραφή πόρων (σ. 126), ο αναλυτής αναγνωρίζει είτε πλήρη URI είτε απόλυτες διαδρομές, για παράδειγμα, `http://example.com/index` και `/index`, αντιστοίχως. Στην περίπτωση πλήρους URI, ο αναλυτής αντιπαραβάλει το όνομα του URI με το όνομα που έχει διευθετηθεί ο διαχομιστής. Το ίδιο συμβαίνει και για τον αριθμό θύρας, εφόσον έχει δηλωθεί στο URI (Berners-Lee, R. Fielding και Masinter, 2005, σ. 49). Κατόπιν, αποπειράται η αναγνώριση του πόρου (μέσω της απόλυτης διαδρομής) και τελικά του ερωτήματος, εφόσον υποστηρίζεται για τον εν λόγω πόρο.

Ανεξαρτήτως πλήρους URI ή διαδρομής, το δηλωθέν URI δύναται να έχει κωδικοποιηθεί με το συμβολισμό % HEX HEX, ο οποίος επιτρέπει τη συμπερίληψη χαρακτήρων που διαφορετικά είναι μη διαθέσιμοι για το URI· χαρακτήρες με ειδικό νόημα εντός των μερών του URI (όπως «@» και «&»), μη εκτυπώσιμοι χαρακτήρες (όπως Αλλαγή Γραμμής – Carriage Return) κ.ά. (Berners-Lee, R. Fielding και Masinter, 1998, σσ. 9–11). Παρότι η υλοποίηση αποφεύγει τη χρήση τέτοιων χαρακτήρων στα URI της, υποστηρίζει την επαναφορά κωδικοποιημένων χαρακτήρων πριν την απόπειρα αναγνώρισης τους.

### 7.2.3 Πόροι διακομιστή

Στον Παγκόσμιο Ιστό, τα URI (Uniform Resource Locator) – ακολουθίες χαρακτήρων δομημένων σύμφωνα με συγκεκριμένους κανόνες σύνταξης – χρησιμοποιούνται για τον προσδιορισμό αφηρημένων ή φυσικών πόρων, προσβάσιμων μέσω του Διαδικτύου ή και μη (Berners-Lee, R. Fielding και Masinter, 2005, σσ. 1,5). Στο πλαίσιο της υλοποίησης, τα URI χρησιμοποιούνται για την αναγνώριση των προιθέσεων της αιτούσας οντότητας σε σχέση με τους διαθέσιμους πόρους (ή υπηρεσίες) της συσκευής. Προκειμένου ο διακομιστής να είναι σε θέση να αποκριθεί καταλλήλως στα εισερχόμενα αιτήματα, είναι απαραίτητο να γνωρίζει τους διαθέσιμους πόρους της συσκευής και τις επιτρεπόμενες ενέργειες επί αυτών. Ιδιαίτερα, η εξέταση και περάτωση κάθε αιτήματος ανατίθεται σε πιο εξειδικευμένα τμήματα του διακομιστή.

Συνολικά, τρία είναι τα βασικά στοιχεία που πρέπει να παρέχονται στο διακομιστή σε ένα πρώτο στάδιο επεξεργασίας των αιτημάτων· το URI, η μέθοδος και, εφόσον ορίζεται, οι παράμετροι του ερωτήματος (query string).

#### Εσωτερική περιγραφή πόρων

Σχετικά με την πρώτη απαίτηση (γνώση των διαθέσιμων πόρων), εφόσον τα URI είναι ακολουθίες χαρακτήρων, αρκεί να τηρείται μία συλλογή αλφαριθμητικών που συγκρίνονται με το URI κάθε αιτήματος ώστε να αποφαίνεται η αποδοχή ή η απόρριψή του. Στο πλαίσιο της υλοποίησης κρίνεται αρκετή η ύπαρξη ενός στατικού συνόλου πόρων και, συνεπώς, μίας αντίστοιχης συλλογής.

Σύμφωνα με τους κανόνες σύνταξης της γραμμής αιτήματος ενός HTTP μηνύματος, ο αναφερόμενος πόρος προσδιορίζεται είτε με πλήρες URI (absoluteURI) είτε με πλήρη διαδρομή (abs\_path) (R. Fielding κ.ά., 1999, σσ. 36–37). Επιλέγεται να υποστηριχθούν οι πιο κοινές μορφές πλήρων URI και διαδρομών, σύμφωνα με τους ακόλουθους κανόνες:

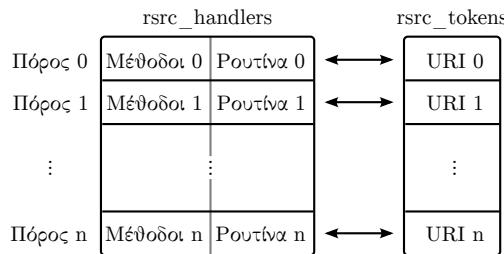
```
absoluteURI = "http://" net_path [ "?" query ]
net_path = "//" authority [ abs_path ]
abs_path = "/" segment *( "/" segment )
authority = host [ ":" port ]
host = hostname | IPv4address
```

Σημειώνεται ότι οι παραπάνω BNF κανόνες αποτελούν μία απλουστευμένη μορφή των κανόνων σύνταξης όπως αυτοί περιγράφονται από τους Berners-Lee, R. Fielding και Masinter (1998, σσ. 11–12, 27–28). Επίσης, θεωρείται ότι ορισμένοι κανόνες (όπως port και IPv4address) είναι περιττό να αναλυθούν περαιτέρω.

Επομένως, παρατηρείται ότι η διαφορά μεταξύ πλήρους URI και πλήρους διαδρομής έγκειται στην ύπαρξη του κειμένου <http://> ακολουθούμενου από το όνομα

#### Σχήμα 7.2.4: Περιγραφή πόρων διαχομιστή.

Κάθε πόρος περιγράφεται στο διαχομιστή HTTP από ένα URI, τις αποδεκτές HTTP μεθόδους και μία αφερωμένη ρουτίνα περάτωσης του αυτήματος.



της αρχής, στην περίπτωση πλήρους URI. Το κομμάτι της αρχής (*authority*) είναι δυνατό να αλλάζει (παρότι σχετικά σπάνια) κατά τη λειτουργία της συσκευής, ενώ, μία δεδομένη στιγμή, είναι ίδια για όλα τα URI. Σε αντίθεση, το *abs\_path* παραμένει σταθερό για κάθε URI. Για τους λόγους αυτούς, χρίνεται προτιμότερη η συμπερίληψη μόνο του τμήματος *abs\_path* στα αλφαριθμητικά της συλλογής, και η εφαρμογή ξεχωριστών επιπρόσθετων ελέγχων σε περίπτωση που παρέχεται πλήρες URI.

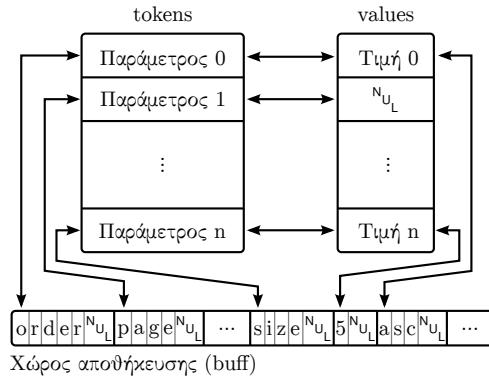
Κάθε πόρος δύναται να υποστηρίζει διαφορετικές μεθόδους και, επομένως, οι διαθέσιμες μέθοδοι πρέπει να προσδιορίζονται για τον καθένα ξεχωριστά. Το ίδιο ισχύει και για τις παραμέτρους ερωτήματος. Ωστόσο, σε αντίθεση με τις μεθόδους, οι παράμετροι έχουν την ιδιαιτερότητα ότι η επιλογή τους γίνεται αυθαίρετα και όχι από ένα σαφώς προσδιορισμένο σύνολο, όπως στην περίπτωση των μεθόδων. Ο τρόπος υποστήριξης παραμέτρων ερωτήματος αναλύονται στη σελίδα 128. Επίσης, αναφέρθηκε ότι η ρουτίνα εξυπηρέτησης των αιτημάτων κάθε πόρου είναι καλύτερο να αποτελεί μία ξεχωριστή διαχειριτή οντότητα. Συνεπώς, χρίνεται χρήσιμος ο ορισμός μίας δομής η οποία παρέχει την πληροφορία URI-μέθοδοι-ρουτίνα για κάθε πόρο.

Ωστόσο, προκειμένου η συλλογή αλφαριθμητικών να είναι συμβατή με τη Λεξική ανάλυση ροής (σ. 118), τα αλφαριθμητικά πρέπει να αποθηκεύονται σε διαδοχικές θέσεις μνήμης, κάτι που ο ορισμός του αλφαριθμητικού εντός της δομής, αποτρέπει. Επομένως, προτιμάται, αρχικά, ο ορισμός της ακόλουθης δομής για την μερική περιγραφή ενός πόρου:

```
typedef struct ResourceHandler {
    uint8_t methods;
    void (*call)(struct HTTPRequest*);
} ResourceHandler;
```

Τυπικά, διατίθεται ένα πλήρως πόρων (και όχι μόνο ένας) και, συνεπώς, απαιτείται

Σχήμα 7.2.5: Δομή αναπαράστασης ερωτήματος URI.



μία συλλογή τέτοιων δομών και μία ακόμα συλλογή με τα αντίστοιχα URI. Η σύνδεση μεταξύ τους γίνεται βάσει της θέσης που κατέχει η κάθε δομή και το URI μέσα στις αντίστοιχες συλλογές, όπως περιγράφεται και στο σχήμα 7.2.4.

### Παράμετροι ερωτήματος

Όπως αναφέρουν οι Berners-Lee, R. Fielding και Masinter (2005, σσ. 23–24) για τις προδιαγραφές του URI, ως ερώτημα (query) ορίζονται μη ιεραρχικά δεδομένα που συνοδεύουν το τμήμα της διαδρομής του URI, η έναρξη των οποίων σηματοδοτείται από το χαρακτήρα «?», και η λήξη, από τον «#» ή, εναλλακτικά, το τέλος του URI, με συνηθισμένη σύνταξη για τη δήλωσή τους, ζεύγη της μορφής «παράμετρος=τιμή». Στο πλαίσιο της υλοποίησης, κρίνεται σκόπιμη η υποστήριξη τους, κατά κύριο λόγο, για την παροχή της αιτούσας οντότητας ενός μηχανισμού για τον έλεγχο των αποτελεσμάτων αναζήτησης.

Τα ερωτήματα έχουν την ιδιαιτερότητα όχι χαρακτηρίζονται από μεγάλη ποικιλομορφία· για κάθε URI είναι δύνατό να υποστηρίζονται αυθαίρετα επιλεγμένα αλφαριθμητικά ως παράμετροι και αντίστοιχων τιμών. Στο πλαίσιο της υλοποίησης, ενδιαφέρει η ύπαρξη μίας υποδομής για την αναγνώριση μόνο εκείνων των παραμέτρων ερωτήματος των εισερχόμενων αιτημάτων που ορίζονται και έχουν νόημα για το εκάστοτε URI, ενώ παράλληλα, παρέχεται ένας χώρος για τη διατήρηση των διοικούντων τιμών με σκοπό τη μετέπειτα επεξεργασία τους από τις αρμόδιες ρουτίνες.

Για το σκοπό αυτό ορίζεται η δομή *QueryString* (σχήμα 7.2.5) όπου κεντρικού ενδιαφέροντος είναι τα μέλη *tokens*, *values* και *buff*. Σε ένα πρώτο στάδιο του κύκλου ζωής της δομής, η συλλογή *tokens* συμπληρώνεται με τις αποδεκτές παραμέτρους του τρέχοντος URI (κατόπιν αναγνώρισής του), ενώ τα στοιχεία της

values αρχικοποιούνται σε NULL υποδηλώνοντας ότι η τιμή όλων των παραμέτρων είναι αόριστη.

Κατά τη διάρκεια ανάλυσης του υπολειπόμενου URI, οι εισερχόμενοι χαρακτήρες συγκρίνονται με τις διαθέσιμες παραμέτρους (εφόσον υπάρχουν), προκειμένου να εντοπιστεί κάποιο ταίριασμα, στην οποία περίπτωση οι εισερχόμενοι χαρακτήρες που αντιστοιχούν στην τιμή (δηλαδή, μετά το =), αντιγράφονται στο χώρο buff και ενημερώνεται ακολούθως η συλλογή values ώστε να καταδειχνύεται το αντίστοιχο αλφαριθμητικό της εντοπισμένης παραμέτρου.

Με το πέρας της ανάλυσης του URI, κάθε στοιχείο της συλλογής values είτε έχει παραμείνει NULL είτε διαθέτει αναφορά σε κάποια θέση του buff από την οποία εκτείνεται ένα αλφαριθμητικό (τερματισμένο με το χαρακτήρα NULL). Η σημασιολογική, πλέον, ανάλυση των τιμών ανατίθεται στην αντίστοιχη ρουτίνα περάτωσης τους αιτήματος μετά την ολοκλήρωση της ανάλυσης και του υπόλοιπου αιτήματος.

#### 7.2.4 Μονάδα αναπαράστασης πόρων

Οι ρουτίνες περάτωσης ενεργοποιούνται για την εξυπηρέτηση των αιτημάτων που απευθύνονται σε κάποιο συγκεκριμένο πόρο (URI). Στο πλαίσιο της υλοποίησης, ενδέχεται το σώμα του αιτήματος να παρέχει την αναπαράσταση της επιδιωκόμενης κατάστασης στην οποία είναι επιθυμητό να τεθεί ο σχετικός πόρος. Οι επιλεγόμενες μορφές αναπαράστασης παρέχουν έναν τρόπο δόμησης (ή οργάνωσης) των δεδομένων με τρόπο που διευκολύνει την αυτοματοποιημένη επεξεργασία τους από διαφορετικές τεχνολογίες και ανεξαρτησία μεταξύ της εσωτερικής υλοποίησης των συμβεβλημένων οντοτήτων (R. T. Fielding, 2000, σσ. 90–92). Η αναπαράσταση αποτελεί, απλώς, το κοινώς αποδεκτό λεξιλόγιο ανταλλαγής των δεδομένων.

Η δόμηση των δεδομένων γίνεται με ζεύγη κλειδιού-τιμής όπου το κάθε κλειδί πληροφορεί τη δομή και τη σημασία της τιμής και επιβάλλεται η χρήση καταχωρημένων μορφών (MIME type) (R. T. Fielding, 2000, σσ. 90–92). Ορισμένες τέτοιες μορφές είναι η XML (Extensible Markup Language) και JSON (Javascript Object Notation).

Ανεξαρτήτως της συγκεκριμένης μορφής, τα ζεύγη κλειδιών-τιμής, ή παράμετροι, παρέχονται σειριακοποιημένα, δηλαδή ως μία ακολουθία χαρακτήρων κειμένου. Προκειμένου να αξιοποιηθούν, οι σειριακοποιημένες παράμετροι πρέπει πρώτα να μετατραπούν σε μορφές οικείες του μικροελεγκτή (όπως αριθμοί και αλφαριθμητικά) από ξεχωριστές μονάδες ειδικά σχεδιασμένες για την ανάλυση δεδομένων μίας συγκεκριμένης μορφής οργάνωσης, τους αναλυτές. Σε εσωτερικό επίπεδο, στο ένα άκρο, οι αναλυτές – γνώστες της σύνταξης μίας δεδομένης αναπαράστα-

σης – δύνανται να αποκτούν πρόσβαση στα υποκείμενα δεδομένα που απαιτούνται από τις ρουτίνες περάτωσης οι οποίες βρίσκονται στο άλλο άκρο.

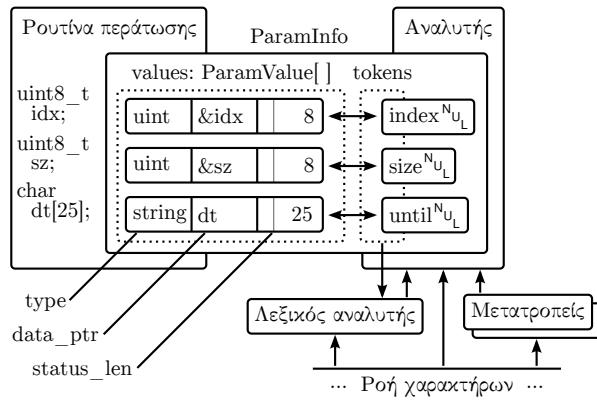
### Αναγωγή παραμέτρων

Λαμβάνοντας υπόψη τους περιορισμένους πόρους του μικροελεγκτή καθώς και προηγούμενες υποδομές, η προσέγγιση που ακολουθείται στο πλαίσιο της υλοποίησης αποτελεί μία επέκταση του τρόπου λειτουργίας των Παραμέτρων ερωτήματος (σ. 128). Σύμφωνα με αυτήν, οι ρουτίνες προετοιμάζουν τις μεταβλητές που πρόκειται να λάβουν τιμή από τη σειριακοποιημένη είσοδο καθώς και ένα πλήθος προσθετών περιγραφών για κάθε μεταβλητή. Ο λόγος ύπαρξης των περιγραφών είναι η πρόκληση της επιθυμητής συμπεριφοράς των αναλυτών απέναντι στην αναμενόμενη είσοδο για την αναγωγή της τιμής κάθε μεταβλητής σύμφωνα με τη σημαντική (ή σημασιολογία) που συνοδεύει κάθε παράμετρο. Συνοπτικά, οι περιγραφές παρέχουν τα ακόλουθα για κάθε μεταβλητή:

- Ένα λεξικό σύμβολο (token), που επιτρέπει την αναγνώριση ύπαρξης της αντίστοιχης παραμέτρου στην είσοδο.
- Τον τύπο δεδομένων, ώστε ο αναλυτής να αναθέτει τη μετατροπή του τιμήματος της εισόδου που αντιστοιχεί στην τιμή, σε κατάλληλη εξωτερική, ανεξάρτητη του αναλυτή, μονάδα: οι μονάδες μετατροπής δύνανται να χρησιμοποιούνται από όλους τους αναλυτές και άλλα υποσυστήματα.
- Το χώρο αποθήκευσης των δεδομένων που ανάγονται από το μετατροπέα. Τυπικά, πρόκειται για τη διεύθυνση της αντίστοιχης μεταβλητής της ρουτίνας περάτωσης.
- Τον προσδιορισμού πρόσθετων ρυθμίσεων για τον αναλυτή (όπως, μέγιστο αναμενόμενο μέγεθος δεδομένων) καθώς και την αναγγελία του αποτελέσματος της επεξεργασίας της παραμέτρου από τον αναλυτή (όπως, ότι η τρέχουσα παράμετρος έχει έγκυρη τιμή).

Το αναμενόμενο είναι ο ορισμός μίας δομής που περικλείει όλες τις παραπάνω πληροφορίες. Ωστόσο, γνωρίζοντας τον υποκείμενο μηχανισμό λεξικής ανάλυσης ροής χαρακτήρων (σ. 118), η δομή `ParamValue` τις περικλείει όλες εκτός από την πρώτη (λεξικό σύμβολο – token). Ο λόγος είναι ότι ο μηχανισμός λεξικής ανάλυσης λειτουργεί με συλλογή αλφαριθμητικών που βρίσκονται σε διαδοχικές θέσεις (για την ακρίβεια, έναν πίνακα διευθύνσεων μνήμης όπου καθεμία αντίστοιχεί στον πρώτο χαρακτήρα αλφαριθμητικών τερματισμένων με NULL). Η συμπερίληψη του κάθε αλφαριθμητικού σε μία πιο σύνθετη δομή, όπως η `ParamValue` όχι τα καθι-

Σχήμα 7.2.6: Διασύνδεση ρουτίνας περάτωσης και αναλυτή αναπαράστασης.



στούσε μη συμβατά με το μηχανισμό ανάλυσης και, ενδεχομένως, θα απαιτούσε τη δημιουργία επικουρικών δομών για την υποστήριξή τους.

Δομή `ParamValue` ορίζεται για κάθε αποδεκτή παράμετρο και διαθέτει τα μέλη `type` (τύπος δεδομένων), `data_ptr` (χώρος αποθήκευσης τιμής) και `status_len` (πρόσθετες ρυθμίσεις και ανατροφοδότηση από αναλυτή). Η δομή `ParamInfo` περιγράφει όλες τις παραμέτρους που διαχειρίζεται μία ρουτίνα περάτωσης και διαθέτει τα μέλη `values` (συλλογή `ParamValue`), `tokens` (συλλογή των αντίστοιχων λεξικών συμβόλων) και `len` (το πλήθος των παραμέτρων).

Το σχήμα 7.2.6 απεικονίζει τη συνήθη χρήση των δομών για τη γνωστοποίηση των αναμενόμενων παραμέτρων στον εκάστοτε αναλυτή. Το μέλος `status_len` προορίζεται για την αναγγελία της μέγιστης επιτρεπτής τιμής που επιτρέπεται να αποθηκευτεί στο `data_ptr` και εξαρτάται από το `type`. Στην περίπτωση που η παράμετρος δέχεται αριθμητικά δεδομένα, το μέγεθος δηλώνει το πλήθος των bit του αποθηκευτικού χώρου `data_ptr`, ενώ στην περίπτωση αλφαριθμητικού, το μέγιστο πλήθος χαρακτήρων.

Ωστόσο, η σημασία του `status_len` είναι διπλή: ένα μέρος του – τα 6 λιγότερο σημαντικά bit – αφιερώνεται για το μέγεθος της τιμής, ενώ ένα άλλο – τα δύο πλέον σημαντικά – για την κατάσταση της παραμέτρου. Υπάρχουν τέσσερις καταστάσεις που καθορίζουν για κάθε παράμετρο εάν έχει τεθεί έγκυρη τιμή ή καθόλου και εφόσον η τιμή είναι εσφαλμένη, εάν το σφάλμα οφείλεται σε λάθος τύπο δεδομένων ή μεγαλύτερη τιμή της επιτρεπόμενη (παραδείγματα, η ακολουθία 102a με δηλωμένο αριθμητικό τύπο ή ο αριθμός 258 με δηλωμένο μήκος 8-bit, αντιστοίχως).

### Αναλυτής αναπαράστασης JSON

Η μοναδική υποστηριζόμενη αναπαράσταση πόρων, στο πλαίσιο της υλοποίησης, επιλέγεται να είναι η JSON (Javascript Object Notation). Στους λόγους επιλογής της έναντι άλλων, συγχαταλέγεται η μικρή επιβάρυνση που επιφέρει στο μέγεθος των σειριακοποιημένων δεδομένων, η σχετικά απλή σύνταξη και, κατά κύριο λόγο, η ευκολία μετατροπής των αναπαραστάσεων σε αντικείμενα σε περιβάλλοντα εκτέλεσης εντός λογισμικών πλοήγησης για τα οποία κυρίως προορίζονται.

Η JSON ορίζει έξι τύπους δεδομένων· αντικείμενο (συλλογή μη ακολουθιακών ζευγών κλειδιού-τιμής), πίνακας (ακολουθία τιμών), αλφαριθμητικό, αριθμός, boolean (true ή false) και null καθώς και έξι δομικούς χαρακτήρες, τους ακόλουθους (T. Bray, 2014, σσ. 3–5):

```
begin-array = ws %x5B ws ; [ Left square bracket
begin-object = ws %x7B ws ; { Left curly bracket
end-array = ws %x5D ws ; ] Right square bracket
end-object = ws %x7D ws ; } Right curly bracket
name-separator = ws %x3A ws ; : Colon
value-separator = ws %x2C ws ; , Comma
ws = *( %x20 / ; Space
        %x09 / ; Horizontal tab
        %x0A / ; Line feed or New line
        %x0D ) ; Carriage return
```

Σημειώνεται ότι η άνω κάτω τελεία (name-separator) χρησιμοποιείται μεταξύ κλειδιού και τιμής στα ζεύγη κλειδιών-τιμές των αντικειμένων, ενώ το κόμμα (value-separator), μεταξύ διαδοχικών τιμών σε αντικείμενα και πίνακες (T. Bray, 2014, σ. 6).

Σημειώνεται ότι η υποστήριξη του συμβολισμού JSON υλοποιείται μόλις σε στοιχειώδη βαθμό.

Στο κομμάτι της ανάλυσης των εισερχόμενων κειμένων JSON, επιλέγεται η υποστήριξη μόνο των άμεσα ενδιαφερόμενων τύπων δεδομένων· αντικείμενο, αριθμός και αλφαριθμητικό. Επιπλέον, η υποστήριξη είναι μερική και υπόκειται στους ακόλουθους περιορισμούς:

- Το κείμενο JSON ορίζει υποχρεωτικά ένα αντικείμενο το οποίο διαθέτει καθόλου ή περισσότερα ζεύγη κλειδιών-τιμές.
- Ένα αντικείμενο διαθέτει ως τιμές μόνο αριθμούς ή αλφαριθμητικά.
- Οι αριθμοί είναι ακέραιοι θετικοί.
- Τα αλφαριθμητικά αποτελούνται από τους χαρακτήρες της κωδικοποίησης Unicode που είναι κοινοί με της κωδικοποίησης ASCII, χωρίς να υπάρχει

υποστήριξη διαφυγής χαρακτήρων (escape character).

Επιπλέον των περιορισμών σχετικά με τους τύπους δεδομένων, ο αναλυτής τερματίζει την επεξεργασία του εισερχόμενου κειμένου με τον εντοπισμό οποιουδήποτε σφάλματος. Ωστόσο, η ύπαρξη ενός κλειδιού πάνω από μία φορά διατηρεί μόνο την τελευταία τιμή.

Στο κομμάτι σειριακοποίησης, δηλαδή της δημιουργίας κειμένου JSON από ένα πλήθος παραμέτρων, η έξοδος υπέκειται στους ίδιους περιορισμούς· δημιουργία ενός αντικειμένου που περιέχει ως τιμές μόνο αριθμούς και αλφαριθμητικά. Ωστόσο, ο αρχικός σχεδιασμός αποτυγχάνει να προβλέψει την ανάγκη για τη χρήση πινάκων ως τιμή κάποιου μέλους του αντικειμένου· κάτι που απαιτείται κατά την επιστροφή αποτελεσμάτων αναζήτησης.

Η αντιμετώπιση παρέχεται με την προαιρετική σταδιακή σειριακοποίηση ορισμένων, κάθε φορά παραμέτρων, προσδιορίζοντας πρόσθετες οδηγίες στο μηχανισμό σειριακοποίησης. Οι οδηγίες αυτές αποτελούν το κριτήριο για την περιγραφή της κατάστασης της εξόδου, όπως αυτή έχει διαμορφωθεί από προηγούμενες ενεργοποιήσεις του μηχανισμού. Με αυτόν τον τρόπο, είναι δυνατό να δημιουργηθεί οποιαδήποτε δομή εξόδου. Ωστόσο, το μεγάλο μειονέκτημα αυτής της προσέγγισης είναι ότι η παραχολούμηση της κατάστασης της εξόδου επαφίεται σε εξωτερική μονάδα και όχι στον ίδιο το μηχανισμό σειριακοποίησης.

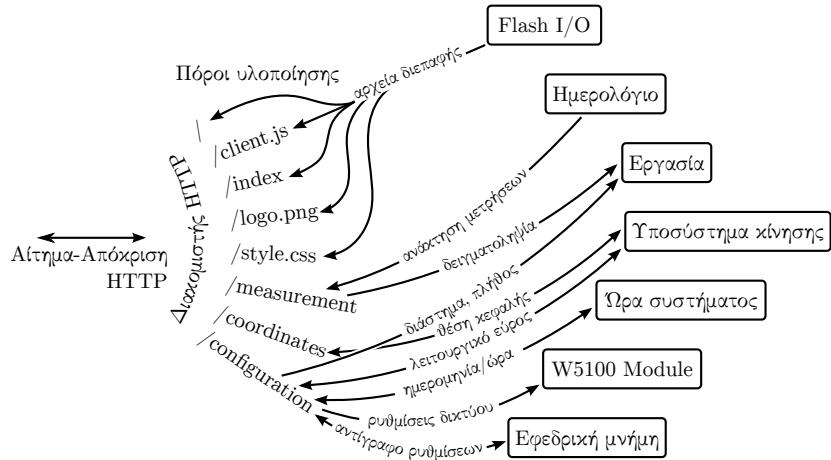
### 7.3 Πόροι υλοποίησης

Οι βασικές αρχές λειτουργίας του διακομιστή περιγράφονται στις προηγούμενες ενότητες. Στο σημείο αυτό ακολουθεί μία περιγραφή των πόρων που διαθέτει η υλοποίηση (σχήμα 7.3.1).

Συνολικά, οι χρησιμοποιούμενες μέθοδοι HTTP είναι η GET, PUT και POST. Η GET χρησιμοποιείται για την ανάκτηση αναπαράστασης κάποιου ή κάποιων πόρων και το URI της, ενδεχομένως, συνοδεύεται από παραμέτρους ερωτήματος (query string). Οι PUT και POST χρησιμοποιούνται για την καταχώρηση της αποστελλόμενης, με το αίτημα, αναπαράστασης σε πόρο του διακομιστή. Η διαφορά τους έγκειται στο ότι η αναπαράσταση σε αίτημα PUT προορίζεται να αποθηκευτεί στον πόρο που προσδιορίζει το URI (ενημερώνοντας τον προϋπάρχοντα ή δημιουργώντας νέο) ενώ σε αίτημα POST, το URI αποτελεί τον πόρο που διαχειρίζεται την αναπαράσταση και, ενδεχομένως, δημιουργεί νέο πόρο υφιστάμενο του δηλωμένου URI (R. Fielding κ.ά., 1999, σσ. 54–55).

Οι λεπτομέρειες εφαρμογής κάθε μεθόδου εξαρτώνται από τον εκάστοτε πόρο. Το ίδιο ισχύει για τη μορφή (media type) της επιστρεφόμενης αναπαράστασης. Όπως έχει αναφερθεί, η υλοποίηση υποστηρίζει τη χρήση JSON ως μορφή ανα-

Σχήμα 7.3.1: Πόροι υλοποίησης και οι μονάδες που χρησιμοποιούν.



παράστασης των πόρων που προορίζονται για επεξεργασία από τρίτα συστήματα. Εκτός από αυτή, χρησιμοποιούνται και οι μορφές (text/html), (text/css), (text/javascript) και (image/png) για τη σύνθεση της διεπαφής που επιτρέπει το χειρισμό της συσκευής δια μέσω λογισμικού πλοιόγησης. Σε κάθε περίπτωση, η μορφή αναπαράστασης (είτε πρόκειται για αίτημα, είτε για απόκρισης) αναφέρεται στο πεδίο κεφαλίδας Content-Type (R. Fielding κ.ά., 1999, σ. 124).

### 7.3.1 Πόρος /configuration

Ο πόρος /configuration αντιπροσωπεύει τις ρυθμίσεις που ορίζουν τη συμπεριφορά της συσκευής.

#### Μέθοδος GET

Η ανάκτηση των τρεχουσών ρυθμίσεων γίνεται με τη χρήση της μεθόδου HTTP GET και επιστρέφεται πάντα κωδικός κατάστασης 200 OK. Η αναπαράσταση είναι η ακόλουθη:

```
{
  "date" : ISO 8601 string,
  "day" : number,
  "gateway" : string,
  "iaddr" : string,
  "interval" : number,
  "samples" : number,
  "subnet" : string,
  "x" : number,
  "y" : number,
```

```

    "z" : number
}

```

**date** Η τρέχουσα ημερομηνία και ώρα της συσκευής (όπως ορίζεται από το RTC).

Επιστρέφεται στην πλήρη μορφή ISO 8601: YYYY-MM-DDTHH:mm:ss.sssZ.

Οστόσο, το κλάσμα δευτερολέπτων είναι πάντα 000. Η συσκευή θεωρεί ότι λειτουργεί σε ζώνη UTC (Universal Time Coordinated).

**day** Η ημέρα της εβδομάδας του RTC. Η τιμή 1 υποδηλώνει «Κυριακή». Η διατήρηση έγκυρης τιμής βασίζεται στη σωστή ρύθμιση της συσκευής.

**gateway** Η διεύθυνση IP της προεπιλεγμένης πύλης.

**iaddr** Η διεύθυνση IP της συσκευής.

**interval** Τα ελάχιστα κβάντα που πρέπει να παρέλθουν προκειμένου να εκκινηθεί νέος κύκλος μετρήσεων. Ένα κβάντο αντιστοιχεί σε 6 λεπτά. Επομένως, οι αποδεκτές τιμές είναι από 0 έως και 240. Η τιμή 0 απενεργοποιεί τις αυτόματες μετρήσεις.

**samples** Το πλήθος των μετρήσεων που πραγματοποιούνται σε κάθε κύκλο. Η τιμή 0, απενεργοποιεί τις αυτόματες μετρήσεις.

**subnet** Η μάσκα υποδικτύου.

**x, y, z** Το προσαρμοσμένο εύρος (ή διαστάσεις) λειτουργίας. Προσδιορίζουν το πλήθος διακριτών θέσεων της κεφαλής ανά άξονα στις οποίες τίθεται προκειμένου να πραγματοποιηθούν μετρήσεις.

Σημειώνεται ότι οι τιμές αυτές υποδηλώνουν το πλήθος των θέσεων σε κάθε άξονα, ενώ οι συντεταγμένες ορίζονται με βάση το 0.

## Μέθοδος PUT

Η τροποποίηση των ρυθμίσεων της συσκευής γίνεται με χρήση της μεθόδου HTTP PUT. Το σώμα του αιτήματος φέρει τις νέες ρυθμίσεων σε αναπαράσταση JSON και μπορεί να περιλαμβάνει μόνο εκείνες τις παραμέτρους που είναι επιθυμητό να τροποποιηθούν. Οι παράμετροι είναι κοινές με την περίπτωση ανάκτησης των ρυθμίσεων (μέθοδος GET).

Οι πιθανοί κωδικοί κατάστασης (status code) της απόκρισης είναι οι ακόλουθοι:

**200 OK** Οι ρυθμίσεις έχουν ενημερωθεί. Σε περίπτωση που δηλώθηκε νέο λειτουργικό εύρος, η κεφαλή επαναφέρεται στις συντεταγμένες {0, 0}. Επιπλέον, όσες μετρήσεις έχουν καταχωρηθεί σε ημερομηνία μεταγενέστερη της date, διαγράφονται (βλ. Επίπεδο κυλιόμενου πίνακα σ. 28).

Το σώμα της απόχρισης περιέχει τις νέες ρυθμίσεις.

**400 Bad Request** Το αίτημα απορρίφθηκε λόγω εσφαλμένης παραμέτρου και/ή τιμής. Το σώμα της απόχρισης περιέχει τις μέγιστες αποδεκτές τιμές για τις παραμέτρους  $x$ ,  $y$ ,  $z$  και  $interval$ , ανεξαρτήτως της φύσης του προβλήματος. Σημειώνεται ότι το αίτημα απορρίπτεται ακόμα και μόνο μία παράμετρος παρουσιάζει σφάλμα.

Για την τροποποίηση των `date` και `day` απαιτείται η ύπαρξη και των δύο στο αίτημα. Διαφορετικά, η παράμετρος αγνοείται. Επίσης, οι ημερομηνία και ώρα παρέχεται στην πλήρη της μορφή (με ή χωρίς κλάσμα δευτερολέπτων).

### 7.3.2 Πόρος /coordinates

Ο πόρος /coordinates αντιπροσωπεύει τη θέση της κεφαλής και επιτρέπει την ανάγνωση της θέσης της καθώς και τη μετακίνησή της σε νέα.

#### Μέθοδος GET

Η ανάκτηση της θέσης της κεφαλής γίνεται με τη χρήση της μεθόδου HTTP GET. Οι πιθανοί κωδικοί κατάστασης (status code) της απόχρισης είναι οι ακόλουθοι:

**200 OK** Το σώμα της απόχρισης περιέχει την αναπαράστασης των συντεταγμένων της κεφαλής σε μορφή JSON. Η δομή είναι η ακόλουθη:

```
{
  "x" : number,
  "y" : number,
  "z" : number
}
```

Η τιμή 0 στην παράμετρο  $z$  υποδηλώνει ότι η κεφαλή βρίσκεται σε κατάσταση λήψης μέτρησης.

**503 Service Unavailable** Η συσκευή είναι απασχολημένη με τη διεκπεραίωση κάποιας άλλης εργασίας (για παράδειγμα, μετατοπίζεται). Το πεδίο κεφαλίδας `Retry-After` δηλώνει τον αναμενόμενο χρόνο ολοκλήρωσης της τρέχουσας εργασίας, σε δευτερόλεπτα.

#### Μέθοδος PUT

Η μετακίνηση της κεφαλής σε νέα θέση γίνεται με χρήση της μεθόδου HTTP PUT. Το σώμα του αιτήματος φέρει το επιμυητό ζεύγος συντεταγμένων στην ακόλουθη μορφή:

```
{
  "x" : number,
  "y" : number
}
```

Οι πιθανοί κωδικοί κατάστασης (status code) της απόχρισης είναι οι ακόλουθοι:

**200 OK** Η συσκευή βρίσκεται ήδη στην καθορισμένη θέση και, συνεπώς, παραμένει εκεί. Το σώμα της απόχρισης περιέχει τις τρέχουσες συντεταγμένες. Η δομή είναι ίδια με την περίπτωση της μεθόδου GET (200 OK).

**202 Accepted** Το αίτημα ήταν έγκυρο και οι καθορισμένες συντεταγμένες έχουν δρομολογηθεί στο υποσύστημα κινητήρων. Το πεδίο κεφαλίδας Retry-After δηλώνει τον αναμενόμενο χρόνο ολοκλήρωσης της μετακίνησης σε δευτερόλεπτα.

**400 Bad Request** Το αίτημα απορρίφθηκε λόγω εσφαλμένης παραμέτρου και/ή τιμής. Το σώμα της απόχρισης περιέχει τις μέγιστες αποδεκτές τιμές για τις παραμέτρους x, y, z, όπως αυτές καθορίζονται από τις ρυθμίσεις της συσκευής (βλ. Μέθοδος GET σ. 134).

**503 Service Unavailable** Ότιο με την περίπτωση της μεθόδου GET (503 Service Unavailable).

### 7.3.3 Πόρος /measurement

Ο πόρος /measurement αντιπροσωπεύει τις μετρήσεις της συσκευής και επιτρέπει την ανάγνωση τους καθώς και την καταχώρηση νέας.

#### Μέθοδος GET

Ανακτά τις καταχωρημένες εγγραφές μετρήσεων από προεπιλογή, όλες. Οι ακόλουθες παράμετροι ερωτήματος (query string) μπορούν να χρησιμοποιηθούν για τον περιορισμό των αποτελεσμάτων:

```
date-since
date-until
page-index
page-size
```

- Οι ημερομηνίες (date-since και date-until) δίνονται σε πλήρη μορφή ISO 8601: YYYY-MM-DDTHH:mm:ss.sssZ. Ωστόσο, το κλάσμα δευτερολέπτων και η ζώνη ώρας είναι προαιρετικά και πάντα αγνοούνται.

Το αποδεκτό εύρος ημερομηνιών είναι μεταξύ των ετών 2000 έως και 2099.

Οποιαδήποτε εκ των δύο ημερομηνιών (ή και οι δύο) μπορούν να χρησιμοποιηθούν.

- Οι παράμετροι `page-index` και `page-size` δέχονται τιμές από 0 μέχρι και 255.

Η παράμετρος `page-size` χωρίζει το σύνολο των εγγραφών που πληρούν τις προϋποθέσεις των ημερομηνιών, σε ιδεατές σελίδες εγγραφών μεγέθους `page-size` η καθεμία. Κατόπιν, η παράμετρος `page-index` καθορίζει ποια σελίδα εγγραφών είναι επιθυμητό να επιστραφεί. Με τον τρόπο αυτό πραγματοποιείται σελιδοποίηση των εγγραφών.

Προφανώς, η παράμετρος `page-index` έχει νόημα και λαμβάνεται υπόψη, μόνο στην περίπτωση που έχει δηλωθεί η `page-size` με τιμή μεγαλύτερη από 0. Εφόσον η `page-index` παραληφθεί, εννοείται τιμή 0, ενώ στην περίπτωση της `page-size`, όλες οι εγγραφές.

Εάν είναι επιθυμητή η επιστροφή μόνο του πλήθους των διαθέσιμων εγγραφών, μπορεί να δηλωθεί `page-size` 0.

Για παράδειγμα, το ακόλουθο URI επιστρέφει το πολύ 10 εγγραφές ξεκινώντας από την 21<sup>η</sup> και που έχουν πραγματοποιηθεί από την 1/1/2015 στις 12:30 και μετά:  
`/measurement?page-size=10&page-index=2&date-since=2015-01-01T12:30:00`

Οι πιθανοί κωδικοί κατάστασης (status code) της απόκρισης σε αιτήματα GET είναι οι ακόλουθοι:

**200 OK** Το σώμα της απόκρισης περιέχει την αναπαράστασης των εγγραφών. Η δομή είναι η ακόλουθη:

```
{
  "page-index": number,
  "page-size" : number,
  "total" : number,
  "log" : [
    {
      "date" : ISO 8601 string,
      "x" : number,
      "y" : number,
      "t" : temperature,
      "ph" : pH,
      "rh" : relative humidity
    },
    ...
  ]
}
```

**page-index, page-size** Η επιστρεφόμενη σελίδα και το μέγιστο πλήθος εγγραφών, όπως δηλώνηκαν στις παραμέτρους ερωτήματος.

**total** Το συνολικό πλήθος των εγγραφών για το προσδιορισμένο εύρος ημερομηνιών, χωρίς την εφαρμογή σελιδοποίησης.

**log** Περιέχει κατά το μέγιστο, page-size εγγραφές. Περιέχεται πάντα στην αναπαράσταση, ακόμα και εάν είναι άδεια από εγγραφές.

Οι εγγραφές αποστέλλονται σε τεμάχια (Παραγωγή τεμαχισμένων μηνυμάτων (σ. 124) ώστε να είναι εγγυημένη η παράδοση οσοδήποτε μεγάλου πλήθους εγγραφών.

**400 Bad Request** Το αίτημα απορρίφθηκε λόγω εσφαλμένης τιμής σε κάποια από τις αποδεκτές παραμέτρους.

### Μέθοδος POST

Επιτρέπει την πραγματοποίηση και καταγραφή μίας μεμονωμένης μέτρησης από την τρέχουσα θέση της κεφαλής. Το σώμα του αιτήματος πρέπει να είναι άδειο. Οι πιθανοί κωδικοί κατάστασης (status code) της απόκρισης είναι οι ακόλουθοι:

**202 Accepted** Η μέτρηση ζεχίνησε. Το πεδίο κεφαλίδας Retry-After δηλώνει τον αναμενόμενο χρόνο ολοκλήρωσης της εργασίας σε δευτερόλεπτα.

**503 Service Unavailable** Η συσκευή είναι απασχολημένη με τη διεκπεραίωση κάποιας άλλης εργασίας (για παράδειγμα, μετατοπίζεται). Το πεδίο κεφαλίδας Retry-After δηλώνει τον αναμενόμενο χρόνο ολοκλήρωσης της τρέχουσας εργασίας, σε δευτερόλεπτα.

#### 7.3.4 Πόροι αρχείων

Τα αρχεία που διανέμει ο διακομιστής, παρότι αφιερώνονται ξεχωριστά URI ώστε να είναι δυνατή η αναγνώρισή τους (για παράδειγμα, /index, /style.css), αναλαμβάνονται από την ίδια ρουτίνα περάτωσης. Για καθένα από αυτά, επιστρέφεται, σαφώς, το κατάλληλο πλήθος Byte από την εξωτερική μνήμη και καταλλήλως συμπληρωμένα πεδία κεφαλίδας (Content-Length, Content-Type και/ή Content-Encoding).

Η μοναδική υποστηριζόμενη μέθοδος HTTP είναι η GET και οι πόροι αρχείων είναι οι ακόλουθοι:

**/, /index** Η βασική διεπαφή της συσκευής (αρχείο HTML).

**style.css** Το αρχείο μορφοποιήσεων της διεπαφής.

**client.js** Κώδικας Javascript για τη διαχείριση της συσκευής δια μέσω της διεπαφής (κάνοντας χρήση των πόρων της συσκευής).

**logo.png** Εικόνα με το λογότυπο του Ιδρύματος.

Σημειώνεται ότι για όλους τους προηγούμενους πόρους εκτός από την εικόνα, η επιστρεφόμενη αναπαράσταση βρίσκεται σε συμπιεσμένη μορφή GZip, το οποίο δηλώνεται μέσω του πεδίου κεφαλίδας «Content-Encoding» (R. Fielding κ.ά., 1999, σσ. 23,118–119). Τα αρχεία αποθηκεύονται εξαρχής σε αυτήν τη μορφή στην εξωτερική μνήμη με κύριο όφελος την εξοικονόμηση αποθηκευτικού χώρου στην αποθήκευση και φορτίου κατά την αποστολή.

## Κεφάλαιο 8

# Συμπεράσματα

Η εργασία παρείχε την ευκαιρία ενασχόλησης με την ανάπτυξης ενός ολοκληρωμένου, μολονότι απλοϊκού, συστήματος αυτοματισμού κάνοντας χρήση σύγχρονων τεχνολογιών, από την αρχή μέχρι το τέλος του.

Ανέδειξε την αξία της ανεξάρτητης μελέτης στην αναζήτηση και αξιολόγηση πηγών και την ανάλυση τεχνικών εγγράφων για την κάλυψη του απαραίτητου γνωστικού υποβάθρου. Έδωσε την ευκαιρία πειραματισμού με γνωστικά πεδία πέραν των ορίων της Πληροφορικής και επέτρεψε τη συλλογή πλούτου εμπειρίας και γνώσεων σε ποικίλους τομείς. Μέσω αυτής επιβεβαιώθηκε, για άλλη μία φορά, η αξία των εφοδίων που παρέχει το Πρόγραμμα Σπουδών και αποτελεί έναυσμα για τη διεύρυνσή τους.

Το αποτέλεσμα της εργασίας πλησιάζει ικανοποιητικά τον προσδοκώμενο στόχο. Παρόλα αυτά, στο υπόλοιπο του κεφαλαίου αναφέρονται ορισμένες ατέλειες που έχουν παρατηρηθεί, εναλλακτικές προσεγγίσεις ορισμένων θεμάτων που θα μπορούσαν να βελτιώσουν το συνολικό αποτέλεσμα καθώς και τρόπους για μία πιθανή επέκταση των δυνατοτήτων του.

### 8.1 Απρόσμενες συμπεριφορές

Για λόγους πληρότητας, αναφέρεται ότι έχουν παρατηρηθεί ορισμένες απρόσμενες συμπεριφορές. Όπως γίνεται αντιληπτό από την περιγραφή τους παρακάτω, και οι τρεις περιπτώσεις παρατηρημένες περιπτώσεις ενδέχεται να είναι προϊόν παραγόντων πέραν των ορίων του λογισμικού που οφείλονται και σε φαινόμενα ηλεκτρονικής ακόμα και μηχανικής φύσεως.

## Αδυναμία επαναφοράς από όριο $Y_{max}$

Μία εξ αυτών σχετίζεται με την κίνηση στον άξονα  $Y$  – μία εκ των διευθύνσεων κίνησης της κεφαλής μετρητών – και, για την ακρίβεια, με την ακινητοποίηση του κινητήρα ως αποτέλεσμα πρόσκρουσης στο τέλος της διαδρομής. Τυπικά, ο μικροελεγκτής αλλάζει το αποστελλόμενο σήμα ώστε ο κινητήρας να οπισθιδρούμησει από το άκρο μέχρι να απελευθερωθεί ο αναστατικός διαχόπτης. Κατόπιν τίθεται σε πορεία που οδηγεί την κεφαλή πίσω στη θέση 0 του άξονα.

Ωστόσο, αυτό που συμβαίνει είναι ότι τη στιγμή που ενεργοποιείται ο διακόπτης, ο κινητήρας συνεχίζει να περιστρέφεται με την ίδια φορά με ασθενέστερο, βέβαια, ρυθμό. Αυτή η συμπεριφορά παρατηρείται κατά την εξώθηση από το μέγιστη θέση και μόνο για τον άξονα  $Y$ .

## Πρόωρη ολοκλήρωση μετατόπισης

Μία δεύτερη ανεξήγητη, μέχρι πρότινος, συμπεριφορά σχετίζεται, εν μέρει, με το υποσύστημα κίνησης και το υποσύστημα διακομιστή HTTP. Εφόσον το πρώτο έχει τεθεί για τη μετατόπιση της κεφαλής και ενώ αυτή βρίσκεται εν κινήσει, εάν ο διακομιστής αφχίσει την επεξεργασία κάποιου εισερχόμενου αιτήματος, ενδέχεται η κίνηση της κεφαλής να ολοκληρωθεί πρόωρα.

Από προσπάθειες απασφαλμάτωσης, ο Χρονομετρητής/Μετρητής που καταμετρά τα βήματα των κινητήρων παρουσιάζεται ότι έχει όντως καταμετρήσει το πλήθυσμα βημάτων που του είχαν δηλωθεί. Ωστόσο, εάν τα δεδομένα που παραλαμβάνονται από το Socket του διακομιστή, απορριφθούν χωρίς να καταναλωθούν (δηλαδή, χωρίς να εισέλθουν) μέσα στο μικροελεγκτή, τότε η μετατόπιση της κεφαλής ολοκληρώνεται πάντα σύμφωνα με το αναμενόμενο.

Σημειώνεται ότι τα εισερχόμενα δεδομένα των αιτημάτων παραμένουν σε προσωρινή μνήμη (buffer) του εξωτερικού ολοκληρωμένου δικτύωσης και υπόκεινται επεξεργασία καθώς αυτά μεταφέρονται στο μικροελεγκτή μέσω διαύλου SPI. Η ακάλειτη ύπαρξη του διαύλου ίσως προκαλούν παρεμβολές στις γραμμές από όπου διέρχονται τα βήματα των κωδικοποιητών, εισάγοντας κίβδηλους παλμούς. Η ιδέα ότι μπορεί να υφίσταται κάτι τέτοιο έχει δούλει από οδηγίες της Maxim (2002, σ. 7) που προειδοποιούν στην περίπτωση όπου ένα ολοκληρωμένο ρολόι πραγματικού χρόνου (RTC) εκτελείται πιο γρήγορα του κανονικού, ενδέχεται να οφείλεται από τη σύζευξη υφορύβου από γειτονικά σήματα στους ακροδέκτες με τους οποίους το RTC συνδέεται με τον εξωτερικό ταλαντωτή.

## Μη ανταπόκριση ολοκληρωμένου δικτύωσης

Μία τρίτη και τελευταία περίπτωση αφορά το ολοκληρωμένο δικτύωσης το οποίο παρατηρήθηκε ότι σε, φαινομενικά, τυχαίες στιγμές, η λυχνία δεδομένων (traffic) που με αυτό συνδέεται, πάλλεται εξακολουθητικά δηλώνοντας ότι είναι μονίμως απασχολημένο, αποτρέποντας τη συσκευή από την εξυπηρέτηση άλλων αιτημάτων.

## 8.2 Βελτιώσεις

### Τποσύστημα κίνησης

Αρκετά είναι τα σημεία που σχετίζονται με την κίνηση της κεφαλής που θα μπορούσαν να δεχθούν βελτιώσεις. Μία σχετίζεται με τους ίδιους τους κινητήρες· επί του παρόντος χρησιμοποιούνται κινητήρες servo οι οποίοι επελέγησαν για ενδεχόμενη περισσότερη ευκολία διασύνδεσης με το μικροελεγκτή. Ωστόσο, εφόσον ενδιαφέρει η ακρίβεια της κίνησης της κεφαλής έναντι της ταχύτητας, ίσως βηματικοί κινητήρες (stepper) να αποβούν καταλληλότεροι.

Η ανάγκη γίνεται ιδιαίτερα αισθητή κατά την μετατόπιση της κεφαλής στο επίπεδο X-Y για το διάστημα που ενεργοποιούνται και οι δύο κινητήρες. Η βασική παραδοχή της υλοποίησης βασίζεται στην υπόθεση ότι σε δεδομένο χρονικό διάστημα, επιτυγχάνεται η ίδια (ή με μικρή απόκλιση) μετατόπιση και στους δύο άξονες (X και Y). Αυτό γίνεται για την αντιμετώπιση του περιορισμένου αριθμού κυκλωμάτων Χρονομετρητών/Απαριθμητών όπου καταμετρούνται τα βήματα μόνο του ενός εκ των δύο κωδικοποιητών κίνησης.

Πιθανές λύσεις είναι αρκετές, μία εξ αυτών είναι η χρήση κάποιου πιο εξελιγμένου μικροελεγκτή ή ενός αφιερωμένου για τον έλεγχο των κινητήρων ελεγκτή (motor controller). Μία πιο απλή λύση υπό τις παρούσες συνθήκες θα ήταν η υποστήριξη μετατόπισης μόνο σε έναν άξονα τη φορά. Τέλος, θα μπορούσε να βελτιωθεί η ακρίβεια (ή διακριτική ικανότητα) των κωδικοποιητών κίνησης όπως, για παράδειγμα, με τη χρήση έτοιμων αντίστοιχων λύσεων.

### Εργασία μετρήσεων

Άλλο σημείο που θα μπορούσε να βελτιωθεί είναι η (αυτόματη) εκκίνηση νέων εργασιών (δηλαδή, κύκλων μετρήσεων). Εάν η συσκευή παραμείνει απενεργοποιημένη για παρατεταμένο χρόνο, όταν τελικά συνδεθεί με την τροφοδοσία ενδέχεται να μην εκκινηθεί νέα μέτρηση αμέσως. Κρίθηκε ότι η συγκεκριμένη συμπεριφορά είναι αποδεκτή για τις ανάγκες της τρέχουσας υλοποίησης. Ωστόσο, θα μπορούσε να τροποποιηθεί ώστε παρακολουθείται ολόκληρη η ημερομηνία αντί απλώς μόνο των ωρών, στις οποίες και οφείλεται αυτή η συμπεριφορά.

Επιπλέον, θα μπορούσε να υλοποιηθεί ένας πιο εκλεπτυσμένος αλγόριθμος επιλογής θέσεων για την πραγματοποίηση μετρήσεων ώστε, για την επιλογή των θέσεων, να λαμβάνεται υπόψη οι θέσεις των πρόσφατα πραγματοποιηθέντων μετρήσεων καθώς και αυτών που είχαν σημειώσει κρίσιμες (οριακές) μετρήσεις.

### Θέματα ισχύος

Επιπλέον, η μείωση της κατανάλωσης ισχύος εφαρμόζεται μόνο στο μικροελεγκτή και το υποσύστημα κίνησης (τόσο για τους κωδικοποιητές όσο και για τους κινητήρες), με διαφόρους τρόπους· ο μικροελεγκτής περνά τον περισσότερο χρόνο του σε κατάσταση power-down, ενώ οι αισθητήρες των κωδικοποιητών (κυρίως του πομπού) διαρρέονται από ρεύμα μόνο κατά τη λειτουργία των κινητήρων.

Θα μπορούσε να εφαρμοστεί και στα υπόλοιπα ολοκληρωμένα κάποια αντίστοιχη τακτική. Για παράδειγμα, η εξωτερική μνήμη Flash διαθέτει λειτουργία χαμηλής κατανάλωσης (Deep power-down mode) στην οποία μπορεί να τεθεί, κατά την οποία η ένταση του ρεύματος που απαιτεί, μειώνεται σε, κατά μέγιστο, 1μΑ από 10μΑ που σημειώνεται σε κατάσταση ετοιμότητας (Microchip, 2003, σσ. 2,16).

### Τυποσύστημα κίνησης

Τέλος, ένα άλλο κοιμάτι το οποίο θα μπορούσε να δεχθεί βελτιώσεις είναι το κοιμάτι του διακομιστή HTTP. Επί του παρόντος, παρότι έχει δοθεί βαρύτητα στην ευελιξία του διακομιστή όσον αφορά τον προγραμματισμό των διατιθέμενων πόρων και τη σύνταξη των πεδίων κεφαλίδας των αποκρίσεων, ο υποκείμενος αναλυτής πεδίων κεφαλίδας καθώς και ο αναλυτής/συντάκτης αναπαραστάσεων JSON έχουν αρκετά σημεία που χρήζουν αναθεώρησης.

## 8.3 Εξέλιξη

Πέραν των πιθανών βελτιώσεων, η υλοποίηση θα μπορούσε να επεκταθεί ώστε να αυτοματοποιεί μεγαλύτερο μέρος της εργασίας. Μία μορφή θα ήταν η υποστήριξη περισσότερων αισθητήρων· παρέχεται ήδη καταμέτρηση της θερμοκρασίας, ωστόσο θα μπορούσε να συμπεριληφθούν η υγρασία και, ενδεχομένως, η οξύτητα του υλικού που, μαζί με τη θερμοκρασία αποτελούν τις βασικές παραμέτρους για τη διατήρηση ενός υγιούς μέσου.

Ένα επόμενο στάδιο, θα ήταν η προσθήκη δυνατοτήτων επενέργειας στο υλικό ώστε, εκτός από παρακολούθηση, η συσκευή να τροποποιεί τις συνθήκες του υλικού. Για παράδειγμα, η ενσωμάτωση ενός εξαρτήματος στη βάση της κεφαλής το

οποίο χρησιμοποιείται για την ύγρανση του υλικού σε περιοχές όπου καταγράφηκε υγρασία κάτω από τα καθορισμένα, από το χρήστη, όρια.

Επιπλέον, θα μπορούσε η ίδια η συσκευή να αναγγέλλει τις καταχωρημένες μετρήσεις της σε τρίτα συστήματα ή και να ειδοποιεί στην περίπτωση που προκύπτει κάποιο χρίσμα συμβάν (alarm), όπως, για παράδειγμα, επικίνδυνα υψηλή θερμοκρασία ή πολλαπλές αστοχίες υλικού (του υποσυστήματος κινητήρων).

Στο κομμάτι της διεπαφής HTTP της συσκευής, θα μπορούσε να εφαρμοστεί διαδικασία διαπίστευσης χρήστη για την προφύλαξη ορισμένων ευαίσθητων λειτουργιών που παρέχει όπως η τροποποίηση των ρυθμίσεών της (για παράδειγμα, διεύθυνση IP). Το συγκεκριμένο παραλήφθηκε, εσκεμμένα, από την τρέχουσα υλοποίηση εξαιτίας των χρονικών περιθωρίων.

Επιπλέον, θα μπορούσε να υποστηριχθούν πιο εύχρηστες ή διαδραστικές μορφές παρουσίασης των μετρήσεων. Στην τρέχουσα κατάσταση, οι μετρήσεις εμφανίζονται σελιδοποιημένες σε πίνακα μεγέθους που επιλέγει ο χρήστης, χωρίς, ωστόσο, να δίνεται η δυνατότητα ταξινόμηση ή να παρέχεται κάποια γραφική απεικόνιση τους. Βέβαια, κάτι τέτοιο θα μπορούσε να υλοποιηθεί από τρίτο σύστημα το οποίο ενδεχομένως συγκεντρώνει στατιστικά στοιχεία και από άλλες παρόμοιες συσκευές.

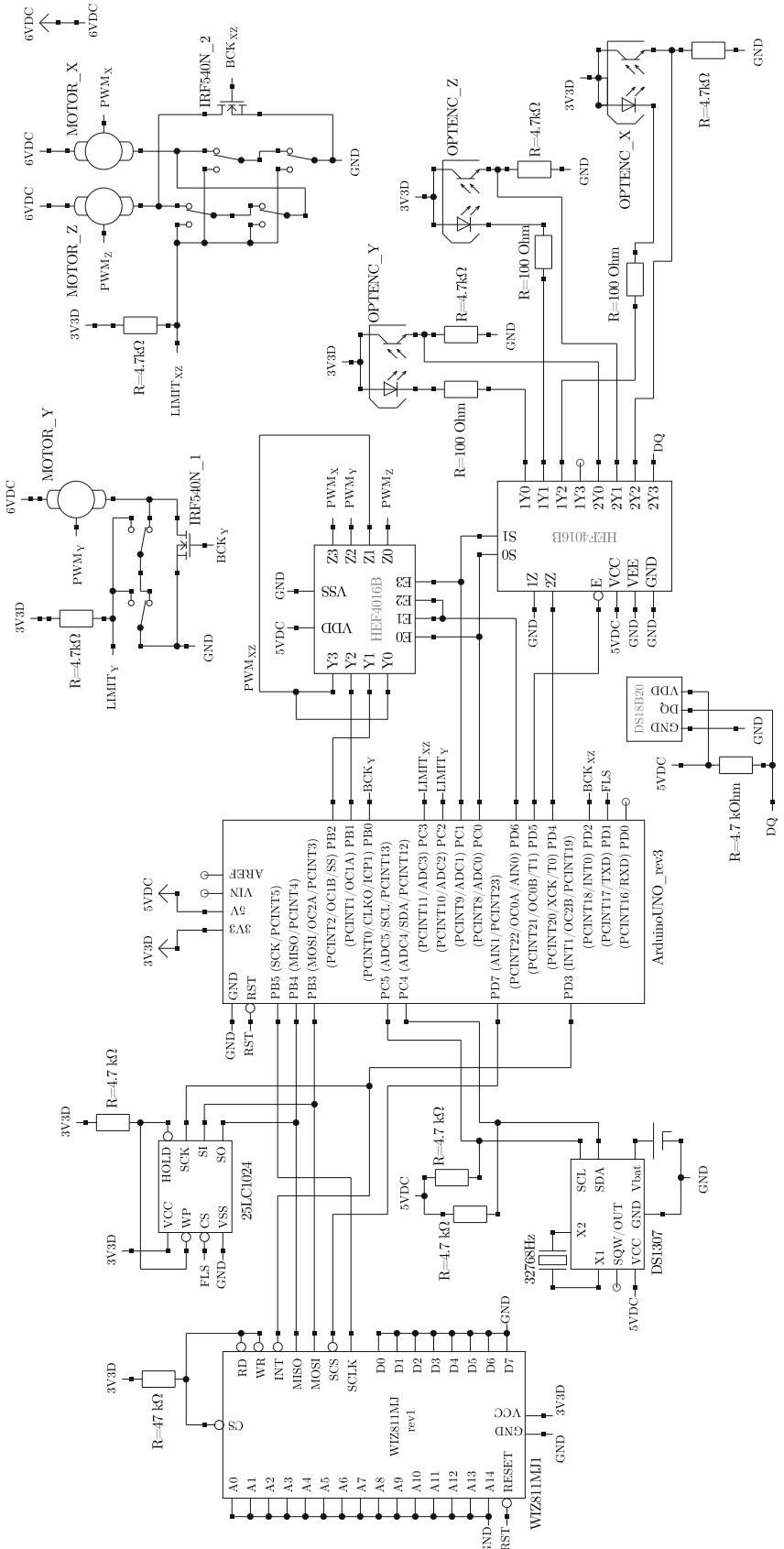


Παραρτήματα



Παράρτημα Α

Συνδεσμολογία



# Παράρτημα Β

## Πηγαίος κώδικας

### B.1 Μικροελεγχτής

Ορισμοί γενικής εφαρμογής

defs.h

```
/**  
 * @file  
 */  
#ifndef DEFS_H_INCL  
#define DEFS_H_INCL  
  
#include <inttypes.h>  
  
/**  
 * @brief Enable communication via the USART.  
 *  
 * Additionally, if @c ENABLE_DEBUG is not defined, provide write-access to the  
 * (external) Flash (from the host).  
 */  
/* #define ENABLE_SERIAL_IO */  
  
/**  
 * @brief Enable transmissions via the USART and disable writing to the Flash.  
 *  
 * This should only be defined, if @c ENABLE_SERIAL_IO is defined as well.  
 */  
/* #define ENABLE_DEBUG */  
  
/**  
 * @brief A coordinate in device space.  
 */
```

```

typedef struct {
    /** @brief X-coordinate (surface abscissa). */
    uint8_t x;

    /** @brief Y-coordinate (surface ordinate). */
    uint8_t y;

    /** @brief Z-coordinate (head elevation). */
    uint8_t z;
} Position;

/**
* @brief Discerns among the available axes.
*
* These constants are not to be OR-ed together.
*/
typedef enum {
    AXIS_X,
    AXIS_Y,
    AXIS_Z
} MotorAxis;

/**
* @brief A date in BCD format.
*/
typedef struct {
    /** @brief Year: @f$0x00–0x99@f$. */
    uint8_t year;

    /** @brief Month: @f$0x01–0x12@f$. */
    uint8_t mon;

    /** @brief Date: @f$0x01–0x31@f$. */
    uint8_t date;

    /** @brief Hours: @f$0x00–0x23@f$. */
    uint8_t hour;

    /** @brief Minutes: @f$0x00–0x59@f$. */
    uint8_t min;

    /** @brief Seconds: @f$0x00–0x59@f$. */
    uint8_t sec;
} BCDDate;

#if defined (ENABLE_DEBUG) && defined (ENABLE_SERIAL_IO)
/**
* @brief Include @p x, if @c ENABLE_DEBUG is specified.

```

```
*  
* @param[in] x Block of code.  
*/  
#define DBG(x) x  
  
#else /* !defined (ENABLE_DEBUG) || !defined (ENABLE_SERIAL_IO) */  
#define DBG(x)  
#endif /* !defined (ENABLE_DEBUG) || !defined (ENABLE_SERIAL_IO) */  
  
/**  
 * @brief Watchdog Time-out.  
 *  
 * The time-out is set to 8s.  
 *  
 * At #WDT_TIMEOUT intervals, the CPU will be woken from power-down mode. The WDT  
 * ISR checks whether sampling should be initiated. Once execution returns to the  
 * main loop, the CPU goes to power-down, again. The CPU maybe woken at any time  
 * by other sources, as well, such as a limit switch and/or an incoming HTTP  
 * request. Those requests could delay the CPU for as long they require with no  
 * fear of resetting the WDT; since the System Reset Mode is not activated, the  
 * WDT Interrupt will simply be queued, if it occurs before any previously  
 * activated ISR returns.  
 */  
#define WDT_TIMEOUT _BV(WDP3) | _BV(WDPO)  
  
/**  
 * @brief Default device IP address.  
 */  
#define FACTORY_IADDR 192, 168, 1, 73  
  
/**  
 * @brief Default device gateway address.  
 */  
#define FACTORY_GATEWAY 192, 168, 1, 1  
  
/**  
 * @brief Default device subnet mask.  
 */  
#define FACTORY_SUBNET 255, 255, 255, 0  
  
/**  
 * @brief Default device hardware address.  
 */  
#define FACTORY_HADDR 0xBE, 0xEB, 0xEE, 0xBE, 0xEB, 0xEE  
  
/**  
 * @brief The size of all backup memory settings.  
 */
```

```
* As these settings are stored in the DS1307 user—memory, no more than 56 Bytes
* may be stored.
*/
#define SYS_SIZE 23

/**
 * @brief User—data RTC memory address.
 *
 * The RTC (DS1307) provides 56 Bytes of user—defined battery—backed RAM. This is
 * the address of the first available byte.
 *
 * Also, see sys_set().
*/
#define RTC_BASE 0x08

/**
 * @brief Backup memory address of IP address.
 */
#define SYS_IADDR (RTC_BASE + 0x00)

/**
 * @brief Backup memory address of Gateway address.
 */
#define SYS_GATEWAY (SYS_IADDR + 0x04)

/**
 * @brief Backup memory address of Subnet mask.
 */
#define SYS_SUBNET (SYS_GATEWAY + 0x04)

/**
 * @brief Backup memory address of Hardware address.
 */
#define SYS_HADDR (SYS_SUBNET + 0x04)

/**
 * @brief Backup memory address of operating range.
 */
#define SYS_MTR_MAX (SYS_HADDR + 0x06)

/**
 * @brief Backup memory address of axis X maximum value.
 */
#define SYS_MTR_MAX_X (SYS_MTR_MAX + 0x00)

/**
 * @brief Backup memory address of axis Y maximum value.
*/
```

```
#define SYS_MTR_MAX_Y (SYS_MTR_MAX_X + 0x01)

/**
 * @brief Backup memory address of axis Z maximum value.
 */
#define SYS_MTR_MAX_Z (SYS_MTR_MAX_Y + 0x01)

/**
 * @brief Backup memory address of automated task settings.
 */
#define SYS_TASK (SYS_MTR_MAX + 0x03)

/**
 * @brief Backup memory address of task intervals.
 */
#define SYS_TASK_INT (SYS_TASK + 0x00)

/**
 * @brief Backup memory address of samples taken at every interval.
 */
#define SYS_TASK_SAMPL (SYS_TASK_INT + 0x01)

/**
 * @brief Get device configuration settings.
 *
 * For a list of available settings, see SYS_* macros. This is provided only for
 * convenience and as a match to sys_set(); it is not necessary to prefer this
 * function over API-specific ones. Also, this function does not access the
 * backup memory.
 *
 * @param[in] setting The system setting to read.
 * @param[out] value The value of @p setting.
 */
void sys_get(uint8_t setting, void* value);

/**
 * @brief Set device configuration settings.
 *
 * The purpose of this function is two-fold; it provides an easy way to pass
 * settings to various modules without the need to use their respective API
 * (which may be too verbose at times); it stores a copy of those settings to a
 * backup memory.
 *
 * For a list of available settings, see SYS_* macros. It should be noted that
 * this is the *preferred* way of updating the settings for which such a macro is
 * provided.
 *
 * The backup memory allows settings persistence even if the device is completely
```

```

* disconnected from the mains. Currently, the backup memory is assigned to the
* battery-backed RTC RAM which allows resetting to factory default settings, if
* the battery is temporarily removed.
*
* @param[in] setting The system setting to update.
* @param[in] value The new value of @p setting.
* @returns @c 0 on success; @c -1, otherwise. Failure designates the setting
* could not be written due to a communication error with the backup memory
* and *not* due to an erroneous value; if the underlying module rejects the
* supplied value, no indication is given by this function. The callee should
* only supply valid values.
*/
int8_t sys_set(uint8_t setting, void* value);

/**
* @brief Frequency of CPU clock, required by <avr/delay.h>.
*/
#define F_CPU (4000000UL)

/**
* @brief USART baud rate.
*
* For the current CPU clock frequency (4MHz), in Asynchronous normal mode, a
* minimal error of 0.2% is achieved at a baud rate of 19.2kbps. *Atmel
* pp.190–193*
*/
#define USART_BAUD (19200)

/**
* @brief Calculates the value for the baud rate register.
*
* For Asynchronous normal mode, the value of UBRR is calculated from the
* following formula: \f[ UBRR = \frac{f_{OSC}}{16 BAUD} - 1 \f] where\n
* \f$ f_{OSC} \f$ is the System Oscillator clock frequency which, in this case is
* down-scaled to 4MHz, and\n
* \f$ BAUD \f$ is the baud rate (in bps).
*/
#define UBRR_VALUE (int)(F_CPU/16/USART_BAUD - 1)

/**
* @brief Value of @c TWPS0 and @c TWPS1 bits of @c TWSR register.
*/
#define TWI_RATE 0

/**
* @brief The prescaler value of the bits specified by #TWI_RATE.
*
* This value is easily determined either by checking the corresponding table

```

```
* (*Atmel p.236*) or, simply, raising @c 4 to the decimal value of #TWI_RATE.  
*  
* Used in calculating #TWBR_VALUE.  
*/  
#define TWI_RATE_VAL 1  
  
/**  
* @brief Frequency of the TWI bus.  
*  
* Currently, the only device on the TWI bus is the RTC DS1307 which operates on  
* the lower TWI scale (up to 100kHz).  
*/  
#define F_TWI (50000UL)  
  
/**  
* @brief Calculates the value for the TWI rate register.  
*  
* *Atmel p.215*  
*/  
#define TWBR_VALUE (int)((F_CPU/F_TWI-16)/2/TWI_RATE_VAL)  
  
/**  
* @brief Socket of W5100 that corresponds to the HTTP server.  
*/  
#define HTTP_SOCKET 0  
  
/**  
* @brief Port the HTTP server is listening to.  
*/  
#define HTTP_PORT 80  
  
/**  
* @brief Available output buffer size in the network module (chip).  
*  
* This is just a convenience. The actual setting is done in code (main()).  
*/  
#define HTTP_BUF_SIZE 2048  
/**  
* @brief Size of the buffer used in parsing query parameters.  
*  
* This much space is allocated upon calling srvr_call() and should be large  
* enough to contain all the acceptable parameters (token + value) for any given  
* resource.  
*  
* It is safe to assume that the server will use this amount of memory only once  
* for a single HTTP request.  
*  
* It is possible to run the server without allocating buffer for such a task
```

```

* (simply by setting this to @c 0).
*/
#define QUERY_BUF_LEN 105

/***
* @brief The maximum number of acceptable parameters for any one resource.
*
* This should be equal to the maximum number of query string parameters that are
* expected by any *one* of the available resource handlers.
*/
#define QUERY_PARAM_LEN 6

/***
* @brief The amount of total records to store in the EEPROM Log.
*
* The available host EEPROM is 1KB, whereas each log record requires 11 bytes
* (see #LogRecord). It is decided to keep tract of the last 90 measurements, so
* the circular buffer requires a total of 990 bytes. From the remainder bytes,
* two more are used; one for #log_index and one for #log_count.
*/
#define LOG_LEN 90

/***
* @brief The EEPROM address to start storing log records.
*
* The circular buffer is stored at the last 990 bytes of the EEPROM. Thus, its
* lowest byte is at this address.
*/
#define LOG_BASE_ADDR 34

/***
* @brief Value of @c SPSR. This should only affect bit @c SPI2X.
*
* Currently, \f$ clk_{IO} \f$ is 4MHz. The 25LC1024 supports transfer rates up
* to 20MHz. The closest that can be attained with the current configuration is
* 2MHz. This requires setting bit @c SPI2X but none of the @c SPR1:0 of @c SPCR.
*/
#define FLS_SPSR _BV(SPI2X)

/***
* @brief Value of @c SPCR. This should only affect bits @c SPCR1:0.
*
* Currently, \f$ clk_{IO} \f$ is 4MHz. The 25LC1024 supports transfer rates up
* to 20MHz. The closest that can be attained with the current configuration is
* 2MHz. This requires setting bit @c SPI2X but none of the @c SPR1:0 of @c SPCR.
*/
#define FLS_SPCR 0

```

```
/**  
 * @brief First flash page address of index.  
  
 * The file starts at this page and extends for #FILE_SIZE_INDEX bytes.  
 * Currently, allocated 15KiB.  
 */  
#define FILE_PAGE_INDEX 32*0  
  
/**  
 * @brief First flash page address of style.css.  
  
 * The file starts at this page and extends for #FILE_SIZE_STYLE_CSS bytes.  
 * Currently, allocated 5KiB.  
 */  
#define FILE_PAGE_STYLE_CSS 32*1  
  
/**  
 * @brief First flash page address of logo.png.  
  
 * The file starts at this page and extends for #FILE_SIZE_LOGO_PNG bytes.  
 * Currently, allocated 5KiB.  
 */  
#define FILE_PAGE_LOGO_PNG (32*1+10)  
  
/**  
 * @brief First flash page address of client.js.  
  
 * The file starts at this page and extends for #FILE_SIZE_CLIENT_JS bytes.  
 * Currently, allocated all the way to the end.  
 */  
#define FILE_PAGE_CLIENT_JS 32*2  
  
/**  
 * @brief Size of the index file.  
 */  
#define FILE_SIZE_INDEX 7899  
  
/**  
 * @brief Size of the style.css file.  
 */  
#define FILE_SIZE_STYLE_CSS 1233  
  
/**  
 * @brief Size of the logo.png file.  
 */  
#define FILE_SIZE_LOGO_PNG 4288  
  
/**
```

```

* @brief Size of the client.js file.
*/
#define FILE_SIZE_CLIENT_JS 6670

/***
* @brief Pulls Flash @c nCS low.
*
* Note that Flash @c nCS is connected to the USART Tx pin.
*/
#define FLS_ENABLE() PORTD &= ~_BV(PORTD1)

/***
* @brief Pulls Flash @c nCS high.
*
* Note that Flash @c nCS is connected to the USART Tx pin.
*/
#define FLS_DISABLE() PORTD |= _BV(PORTD1)

/***
* Value of @c TOP (@c OC1A) that produces pulses at 50Hz taking #MTR_PRESCALER
* into consideration.
*/
#define MTR_TOP (int)(F_CPU/2/8/50)

/***
* @brief Motor PWM prescaler bits.
*
* The prescaler to use for motor PWM generation. Granted \f$ clk_{IO} \f$ is 4MHz (see #F_CPU), a prescaler of \f$ clk_{IO}/8 \f$ to produce a 50Hz signal, provides 5000 discrete steps of resolution. *Atmel pp.133—135.*
*
* The value should be the bitwise-OR operation of the appropriate bits of @c TCCR1B.
*/
#define MTR_PRESCALER (_BV(CS11))

/***
* @brief Data Direction Register of pin X and Z motors connect to.
*
* Also, see #MTR_XZ_PORT and #MTR_XZ.
*/
#define MTR_XZ_DDR DDRB

/***
* @brief The pin port X and Z motors connect to.
*
* Also, see #MTR_XZ_DDR and #MTR_XZ.
*/

```

```
#define MTR_XZ_PORT PORTB

/**
 * @brief Pin of #MTR_XZ_PORT where X and Z motors connect to.
 *
 * This pin is used for the transmission of the PWM signal. Note that in this
 * implementation, both X and Z motors receive signal from the same pin; it is
 * the value of pins #MUX_S0 and #MUX_S1 that determine which one the signal
 * actually reaches to.
 *
 * Also, see #MTR_XZ_DDR and #MTR_XZ_PORT.
 */

#define MTR_XZ PORTB2

/**
 * @brief Data Direction Register of pin Y motor connects to.
 *
 * Also, see #MTR_Y_PORT and #MTR_Y.
 */

#define MTR_Y_DDR DDRB

/**
 * @brief The pin port Y motor connects to.
 *
 * Also, see #MTR_Y_DDR and #MTR_Y.
 */

#define MTR_Y_PORT PORTB

/**
 * @brief Pin of #MTR_Y_PORT where Y motor connects to.
 *
 * This pin is used for the transmission of the PWM signal.
 *
 * Also, see #MTR_Y_DDR and #MTR_Y_PORT.
 */

#define MTR_Y_PORTB1

/**
 * @brief Data Direction Register of pin the AutoLock MOSFET's Gate connects to.
 *
 * Also, see #MTR_nLOCK_PORT and #MTR_nLOCK.
 */

#define MTR_nLOCK_DDR DDRD

/**
 * @brief The pin port the AutoLock MOSFET's Gate connects to.
 *
 * Also, see #MTR_nLOCK_PORT and #MTR_nLOCK.
```

```

*/
#define MTR_nLOCK_PORT PORTD

/**
 * @brief The pin port the AutoLock MOSFET's Gate connects to.
 *
 * Also, see #MTR_nLOCK_PORT and #MTR_nLOCK.
 */
#define MTR_nLOCK_PIN PIND

/**
 * @brief Pin of #MTR_nLOCK_PORT the AutoLock MOSFET's Gate connects to.
 *
 * This pin enables and disables the Lock and, in order for the AutoLock
 * mechanism to operate, it must coincide with pin @c OCOA. It is active low.
 *
 * Also, see #MTR_nLOCK_PORT and #MTR_nLOCK.
 */
#define MTR_nLOCK PORTD6

/**
 * @brief Data Direction Register of pin MUX @c nCS connects to.
 *
 * Also, see #MUX_nCS_PORT and #MUX_nCS.
 */
#define MUX_nCS_DDR DDRD

/**
 * @brief The pin port MUX @c nCS connects to.
 *
 * Also, see #MUX_nCS_DDR and #MUX_nCS.
 */
#define MUX_nCS_PORT PORTD

/**
 * @brief Pin of #MUX_nCS_PORT MUX @c nCS connects to.
 *
 * This pin enables and disables the multiplexer. It is active low.
 *
 * Also, see #MTR_nLOCK_PORT and #MTR_nLOCK.
 */
#define MUX_nCS PORTD5

/**
 * @brief Data Direction Register of pin MUX @c 2Z connects to.
 *
 * Also, see #MUX_2Z_PIN and #MUX_2Z.
 */

```

```
#define MUX_2Z_DDR DDRD

/**
 * @brief The pin port MUX @c 2Z connects to.
 *
 * Also, see #MUX_2Z_DDR and #MUX_2Z.
 */
#define MUX_2Z_PIN PIND

/**
 * @brief Pin of #MUX_2Z_PIN MUX @c 2Z connects to.
 *
 * This pin gives the output of the rotary encoders (step) of motors Y, Z and X
 * on pins @c 2Y0, @c 2Y1 and @c 2Y2 of the multiplexer, respectively. Pin @c 2Y3
 * gives access to the 1-wire DQ line.
 *
 * Also, see #MUX_2Z_PIN and #MUX_2Z_PIN.
 */
#define MUX_2Z PORTD4

/**
 * @brief Data Direction Register of pin MUX @c S0 connects to.
 *
 * Also, see #MUX_S0_PORT and #MUX_S0.
 */
#define MUX_S0_DDR DDRC

/**
 * @brief The pin port MUX S0 connects to.
 *
 * Also, see #MUX_S0_DDR and #MUX_S0.
 */
#define MUX_S0_PORT PORTC

/**
 * @brief Pin of #MUX_S0_PORT MUX @c S0 connects to.
 *
 * This pin along with pin #MUX_S1 select which MUX pin, @c 1Y0, @c 1Y1, @c 1Y2
 * or @c 1Y3, connects to ground. The first three pins enable a rotary encoder
 * (Y, Z or X, respectively) while the last one, the ADC. Apparently, these are
 * all active low. They also select one of MUX @c 2Y0, 2Y1, 2Y2 or 2Y3 as input
 * on pin #MUX_2Z. The first three correspond to the step (feedback) of a rotary
 * encoder (in same order presented above) or, in case of @c 2Y3, the 1-wire DQ
 * line.
 *
 * Motors X and Z receive PWM signal from the same pin, pin #MTR_XZ, while pins
 * #MUX_S0 and #MUX_S1 select which one the PWM signal is actually propagated to.
 * Because this is implemented by two single switches directly connected to
```

```

* #MUX_SO and #MUX_S1, both motors will be receiving any signal on #MTR_XZ when
* selecting the third MUX channel (@c 1Y3, @c 2Y3). It should be noted though
* that in such a case, no PWM signal should even be present on #MTR_XZ (ie,
* while making a conversion with the ADC or the DS18B20 -- the only 1-wire
* device of this implementation). Also, there is #MTR_nLOCK which could further
* disengage PWM transmission.
*
* Also, see #MUX_S1, #MUX_SO_DDR and #MUX_SO_PORT.
*/
#define MUX_SO PORTC0

/**
* @brief Data Direction Register of pin MUX @c S1 connects to.
*
* Also, see #MUX_S1_PORT and #MUX_S1.
*/
#define MUX_S1_DDR DDRC

/**
* @brief The pin port MUX S1 connects to.
*
* Also, see #MUX_S1_DDR and #MUX_S1.
*/
#define MUX_S1_PORT PORTC

/**
* @brief Pin of #MUX_S1_PORT MUX @c S1 connects to.
*
* This pin along with pin #MUX_SO are the select bits of the multiplexer.
* For details, refer to #MUX_SO.
*
* Also, see #MUX_S1_DDR and #MUX_S1_PORT.
*/
#define MUX_S1_PORTC1

/**
* @brief Chip-selects the multiplexer.
*
* The multiplexer enables and gives access to the output (step) of the rotary
* encoders on channels 0 through 2. Channel 3 chip-selects the ADC and grants
* access to the 1-wire DQ line through pin #MUX_2Z. Currently, only the DS18B20
* (digital thermometer) is connected on the DQ line.
*/
#define MUX_ENABLE() MUX_nCS_PORT &= ~_BV(MUX_nCS)

/**
* @brief Disables the multiplexer, setting all its pins on state Z.
*

```

```
* Also, see #MUX_ENABLE().  
*/  
#define MUX_DISABLE() MUX_nCS_PORT |= _BV(MUX_nCS)  
  
/**  
 * @brief Pin Change interrupt vector to enable on a limit switch change.  
 *  
 * This particular value is used to enable pin change interrupts on pins  
 * @c PCINT\[14:8\] (*Atmel pp.74–75*). For simplicity's sake, it is assumed that  
 * limits on all axes use pins of the same *port*.  
 *  
 * Also, see #LMT_PCMSK and LMT_PCMSK_VAL.  
 */  
#define LMT_PCIE PCIE1  
  
/**  
 * @brief PCINT mask register that corresponds to the port defined by #LMT_PCIE.  
 *  
 * Also, see #LMT_PCIE and LMT_PCMSK_VAL.  
 */  
#define LMT_PCMSK PCMSK1  
  
/**  
 * @brief Value of #LMT_PCMSK that enables interrupts on the pins of the limit  
 * switches.  
 *  
 * The ones of interest are @c PCINT11 (@c PC3) and @c PCINT10 (@c PC2).  
 *  
 * Also, see #LMT_PCIE and #LMT_PCMSK.  
 */  
#define LMT_PCMSK_VAL _BV(PCINT11) | _BV(PCINT10);  
  
/**  
 * @brief Data Direction Register of pin Y limit strobe connects to.  
 *  
 * Also, see #LMT_nY_PIN and #LMT_nY.  
 */  
#define LMT_nY_DDR DDRC  
  
/**  
 * @brief The pin port Y limit strobe connects to.  
 *  
 * Also, see #LMT_nY_DDR and #LMT_nY.  
 */  
#define LMT_nY_PIN PINC  
  
/**  
 * @brief Pin of #LMT_nY_PIN Y limit strobe connects to.
```

```

*
* The line is externally pulled high. When one of the Y SPDT limit switches are
* reached, the line is pulled low indicating the incident and remains low until
* the switch is released back to its normally-closed (NC) state. Refer to #BCK_Y
* for details on how this is achieved.
*
* Note this pin will go low when *either* limit switch is reached; to determine
* which one, the direction of motion should be considered.
*
* Also, see #LMT_nY_DDR and #LMT_nY_PIN.
*/
#define LMT_nY PORTC2

/**
* @brief Data Direction Register of pin XZ limit strobes connect to.
*
* Also, see #LMT_nXZ_PIN and #LMT_nXZ.
*/
#define LMT_nXZ_DDR DDRC

/**
* @brief The pin port XZ limit strobes connect to.
*
* Also, see #LMT_nXZ_DDR and #LMT_nXZ.
*/
#define LMT_nXZ_PIN PINC

/**
* @brief Pin of #LMT_nXZ_PIN XZ limit strobes connect to.
*
* The line is externally pulled high. When one of the X or Z SPDT limit switches
* are reached, the line is pulled low indicating the incident and remains low
* until the switch is released back to its normally-closed (NC) state. Refer to
* #BCK_XZ for details on how this is achieved.
*
* Note this pin will go low when *any* limit switch is reached; to determine
* which one, the direction of motion as well as which motor, X or Z, was enabled
* at the time the limit was reached should be considered. Also, note that
* reaching a limit on one axis would automatically disable recognition of limit
* on the other, unless NC state is restored in the former, first. Generally,
* this should not be an issue since motion on both X and Z axes at the same time
* should not take place.
*
* Also, see #LMT_nXZ_DDR and #LMT_nXZ_PIN.
*/
#define LMT_nXZ PORTC3

/**

```

```
* @brief Data Direction Register the Backtrack MOSFET's Gate connects to.  
*  
* Also, see #BCK_XZ_PORT and #BCK_XZ.  
*/  
#define BCK_XZ_DDR DDRD  
  
/**  
* @brief The pin port the Backtrack MOSFET's Gate connects to.  
*  
* Also, see #BCK_XZ_DDR and #BCK_XZ.  
*/  
#define BCK_XZ_PORT PORTD  
  
/**  
* @brief Pin of #BCK_XZ_PORT the Backtrack MOSFET's Gate connects to.  
*  
* This pin controls a secondary connection to ground of motors X and Z. It  
* should normally be pulled low and enabled only when an X or Z limit switch has  
* been engaged. This, along with a reverse PWM signal should move the apparatus  
* away from the switch just enough to disengage it and permit normal operation  
* of the motor.  
*  
* Also, see #BCK_XZ_DDR and #BCK_XZ_PORT.  
*/  
#define BCK_XZ PORTD2  
  
/**  
* @brief Data Direction Register the Backtrack MOSFET's Gate connects to.  
*  
* Also, see #BCK_Y_PORT and #BCK_Y.  
*/  
#define BCK_Y_DDR DDRB  
  
/**  
* @brief The pin port the Backtrack MOSFET's Gate connects to.  
*  
* Also, see #BCK_Y_DDR and #BCK_Y.  
*/  
#define BCK_Y_PORT PORTB  
  
/**  
* @brief Pin of #BCK_Y_PORT the Backtrack MOSFET's Gate connects to.  
*  
* Similar to #BCK_XZ, only this time, it controls a secondary connection to  
* ground for motor Y.  
*  
* Also, see #BCK_Y_DDR and #BCK_Y_PORT.  
*/
```

```

#define BCK_Y PORTB0

/**
 * @brief Pin of #W1_DQ_PORT where 1-wire DQ line connects to.
 *
 * This pin is used for communicating via the 1-wire protocol.
 *
 * Also, see #W1_DQ_PORT, #W1_DQ_DDR and #W1_DQ_PIN.
 */
#define W1_DQ PORTD4

/**
 * @brief The pin port 1-wire DQ connects to.
 *
 * Also, see #W1_DQ, #W1_DQ_DDR and #W1_DQ_PIN.
 */
#define W1_DQ_PORT PORTD

/**
 * @brief Data Direction Register of pin 1-wire DQ connects to.
 *
 * Also, see #W1_DQ_PORT, #W1_DQ_DDR and #W1_DQ_PIN.
 */
#define W1_DQ_DDR DDRD

/**
 * @brief The pin port 1-wire DQ connects to.
 *
 * Also, see #W1_DQ, #W1_DQ_PORT and #W1_DQ_DDR.
 */
#define W1_DQ_PIN PIND

/**
 * @brief MCU pin which the \f$\overline{RESET}\f$ pin of the network controller
 * is connected to.
 *
 * In order to properly reset the W5100 TCP/IP controller, this pin must be held
 * low for at least 2us. *WIZnet p.9.*
 */
#define NET_RST PORTD4

/**
 * @brief Number of bytes dedicated to W5100 I/O.
 */
#define NET_BUF_LEN (100)

/**
 * @brief Value of @c SPCR. This should only affect bits @c SPCR1:0.
 */

```

```
/*
 * Currently, \f$ clk_{IO} \f$ is 4MHz. The W5100 supports transfer rates up
 * to 14MHz. The closest that can be attained with the current configuration is
 * 2MHz. This requires setting bit @c SPI2X but none of the @c SPR1:0 of @c SPCR.
 */
#define NET_SPCR 0

/**
 * @brief Value of @c SPCR. This should only affect bits @c SPCR1:0.
 *
 * Currently, \f$ clk_{IO} \f$ is 4MHz. The W5100 supports transfer rates up
 * to 14MHz. The closest that can be attained with the current configuration is
 * 2MHz. This requires setting bit @c SPI2X but none of the @c SPR1:0 of @c SPCR.
 */
#define NET_SPSR _BV(SPI2X)

/**
 * @brief Data Direction Register the W5100 nCS pin connects to.
 *
 * Also, see #NET_nCS_PORT and #NET_nCS.
 */
#define NET_nCS_DDR DDRD

/**
 * @brief The pin port the W5100 nCS pin connects to.
 *
 * Also, see #NET_nCS_DDR and #NET_nCS.
 */
#define NET_nCS_PORT PORTD

/**
 * @brief Pin of #NET_nCS_PORT the W5100 nCS pin connects to.
 *
 * Also, see #NET_nCS_DDR and #NET_nCS_PORT.
 */
#define NET_nCS_PORTD7

/**
 * @brief Chip-select the W5100.
 */
#define NET_ENABLE() (NET_nCS_PORT &= ~_BV(NET_nCS))

/**
 * @brief Chip-deselect the W5100.
 */
#define NET_DISABLE() (NET_nCS_PORT |= _BV(NET_nCS))

#endif /* DEFS_H_INCL */
```

## Εξωτερική μνήμη Flash

### flash.h

```

/***
* @file
***/

#ifndef FLASH_H_INCL
#define FLASH_H_INCL

#include "defs.h"

#include <inttypes.h>

/***
* @brief Read flash, beginning at the specified address.
*/
#define FLS_READ 0x03

/***
* @brief Write flash, beginning at the specified address.
*/
#define FLS_WRITE 0x02

/***
* @brief Set the write-enable latch.
*/
#define FLS_WREN 0x06

/***
* @brief Reset the write-enable latch.
*/
#define FLS_WRDI 0x04

/***
* @brief Read Status Register.
*/
#define FLS_RDSR 0x05

/***
* @brief Write Status Register.
*/
#define FLS_WRSR 0x01

/***
* @brief Erase page the specified address belongs to.
*/

```

```
#define FLS_PE 0x42

/**
 * @brief Erase sector the specified address belongs to.
 */
#define FLS_SE 0xD8

/**
 * @brief Erase chip.
 */
#define FLS_CE 0xC7

/**
 * @brief Wake from Deep power-down mode and return device signature.
 */
#define FLS_RDID 0xAB

/**
 * @brief Read flash, beginning at the specified address.
 */
#define FLS_DPD 0xB9

/**
 * @brief Status Register bit Write-In-Progress; @c 1 indicates @c true.
 *
 * Read-only.
 */
#define FLS_WIP 0

/**
 * @brief Status Register bit Write-Enable Latch; @c 1 indicates @c enabled.
 *
 * Read-only.
 */
#define FLS_WEL 1

/**
 * @brief Status Register Block Protection bits @c BP1:0.
 *
 * Read/Write.
 *
 * These bits determine which sectors are write-protected. The combinations of
 * @c BP1:0 are as follows:
 * - @c 00: No sectors protected.
 * - @c 01: Protected sector 3 (18000h---1FFFFh)
 * - @c 10: Protected sector 2 and 3 (10000h---1FFFFh)
 * - @c 11: All sectors protected (00000h---1FFFFh)
 *
```

```

* *25LC1024 p.11*
*/
#define FLS_BP0 2

/**
* @brief Status Register Block Protection bits @c BP1:0.
*
* Read/Write.
*
* These bits determine which sectors are write-protected. See #FLS_BP0.
*/
#define FLS_BP1 3

/**
* @brief Status Register bit to controls write-access to nonvolatile bits of SR.
*/
#define FLS_WPEN 7

/**
* @brief Prepare the SPI bus to communicate with the Flash.
*
* This function:
* — Disables SPI (in case it was running).
* — Sets up the appropriate clock rate (see #FLS_SPSR and #FLS_SPCR).
* — Enables the chip (see #FLS_ENABLE()).
* — Delays 1us for @c nCS setup time (T_{CSS}, *25LC1024 p.3*).
* — Does *not* enable the SPI clock!
*/
void fls_select();

/**
* @brief Terminate communication with the Flash.
*
* This function:
* — Delays 1us for @c nCS hold time (T_{CSH}, *25LC1024 p.3*).
* — Disables the chip (see #FLS_DISABLE()).
* — Delays 1us for MISO output disable time (T_{DIS}, *25LC1024 p.4*).
*/
void fls_deselect();

/**
* @brief Busy-wait until any pending write operations have completed.
*/
void fls_wait_WIP();

/**
* @brief Send the specified command, optionally receiving/sending data.
*

```

```
* This function, in terms of the 25LC1024 instruction set, may be used to send a
* command:
* — with no additional payload,
* — followed by a single byte,
* — and read one byte from the chip.
*
* To send/read multiple bytes to/from a specific address range, use
* fls_exchange().
*
* Once completed, the @c nCS will be pulled high.
*
* @param[in] c The command to send.
* @param[in,out] data If not @c NULL, the contents of the address are sent to
* the Flash and the response is written it.
*/
void fls_command(uint8_t c, uint8_t* data);

/**
* @brief Exchange data with the Flash starting at a particular page's byte 0.
*
* It sends command @p c followed by an address (calculated from @p page). It may
* optionally send *and* receive @p len bytes. Note that this function
* *exchanges* bytes and, so, it *always* sends @p len bytes from @p buf and
* writes that many bytes back into it.
*
* Note that this function accepts the page to start reading/writing from/to and
* *not* an address, implying that each exchange begins at the first byte of any
* page. Also note that the 25LC1024 may not be written data on successive pages
* *at the same time*; on the contrary, the data will be wrapped around to the
* beginning of the page, overwriting any contents (*25LC1024 p.6*); a
* limitation/feature of the chip. Using this function, this typically means,
* that no more than 256 bytes should be exchanged for a @c WRITE operation.
*
* @param[in] c The command to send.
* @param[in] page The page to read from or write to. The 25LC1024 contains
* 1Mbit, organised in 8-bit words. Each page contains 256 bytes, so there are
* 512 pages available (@c 0 through @c 511);
* @param[in,out] buf The bytes to sent. Upon return, it contains the bytes read
* from the Flash. Note that the contents of this array are *always* altered,
* even if @c specified an output command!
* @param[in] len Size of bytes to exchange (read/write); @c 0 is valid, in which
* case, @p buf could be @c NULL.
*/
void fls_exchange(uint8_t c, uint16_t page, uint8_t* buf, uint16_t len);

#endif /* FLASH_H_INCL */
```

**flash.c**

```

#include "flash.h"

#include <avr/io.h>

#include <util/delay.h>

void fls_select() {
    /* Disable SPI, if running. */
    SPCR = 0;

    /* Use the specified clock settings in SPI Master mode (0,0). Do *not*
     * enable. */
    SPSR = FLS_SPSR & _BV(SPI2X);
    SPCR = (FLS_SPCR & (_BV(SPR1) | _BV(SPRO))) | _BV(MSTR);

    /* Set the appropriate signals to select the Flash memory (through the mux).
     * Wait for a minimum of 25ns (if V_{CC} is 4.5—5.5) before sending any CLK
     * pulses. (T_{CSS} — @c nCS setup time, *25LC1024 p.3*) */
    FLS_ENABLE();
    _delay_us(1);
}

void fls_deselect() {
    /* Wait before pulling @c nCS high (T_{CSH} 50ns, *25LC1024 p.3*). */
    _delay_us(1);
    FLS_DISABLE();

    /* Wait the appropriate amount of time for Flash to release MISO line after
     * deselecting it. (T_{DIS} — Output disable time, T_{REL} nCS high to
     * Standby mode, *25LC1024 p.3—4*). */
    _delay_us(2);
}

void fls_wait_WIP() {
    uint8_t status;

    do {
        fls_command(FLS_RDSR, &status);
    } while(bit_is_set(status, FLS_WIP));
}

void fls_command(uint8_t c, uint8_t* data) {
    fls_select();
    SPCR |= _BV(SPE);

    SPDR = c;
}

```

```

loop_until_bit_is_set(SPSR, SPIF);

if(data) {
    SPDR = *data;
    loop_until_bit_is_set(SPSR, SPIF);
    *data = SPDR;
}

fls_deselect();
}

void fls_exchange(uint8_t c, uint16_t page, uint8_t* buf, uint16_t len) {
    uint16_t i;
    uint8_t addr[3];

    /* Calculate the starting address of @p page. */
    addr[0] = page >> 8;
    addr[1] = page & 0xFF;
    addr[2] = 0;

    /* Send the command. */
    fls_select();
    SPCR |= _BV(SPE);
    SPDR = c;
    loop_until_bit_is_set(SPSR, SPIF);

    /* Send the address. This part could be merged with the next (sending
     * data).*/
    for(i = 0 ; i < 3; ++i) {
        SPDR = addr[i];
        loop_until_bit_is_set(SPSR, SPIF);
    }

    /* Send data, if any. */
    for(i = 0 ; i < len; ++i) {
        SPDR = buf[i];
        loop_until_bit_is_set(SPSR, SPIF);

        buf[i] = SPDR;
    }
    fls_deselect();
}

```

## Αναλυτής HTTP

### http\_parser.h

```

/***
 * @file

```

```
* @addtogroup http_parser HTTP Parser
* @ingroup http_server
* @{
*/
#ifndef WEB_SERVER_H_INCL
#define WEB_SERVER_H_INCL

#include "http_server.h"
#include "resource.h"

#include <inttypes.h>

/***
* @brief Indicates a CRLF sequence.
*/
#define CRLF -4

#ifndef EOF
/***
* @brief Indicates that the End-of-File has been reached.
*/
#define EOF -1
#endif

/***
* @brief Provide a reference to the HTTP server settings.
*
* The HTTP parser requires access to HTTP tokens (such as header names),
* resource absolute paths, server host name and port in order to analyse
* incoming HTTP messages. In no way does it alter the contents of the supplied
* variable.
*
* @param[in] srvr Pointer to a valid ServerSettings variable.
*/
void http_parser_set_server(ServerSettings* srvr);

/***
* @brief Parse the input stream and return an #HTTPRequest representation of it.
*
* The contents of the input stream are considered to be an HTTP request. First,
* the request line is parsed (see #parse_request_line()). Then, the headers
* follow (see #parse_headers()). Headers that are not supported are ignored.
*
* Generally, the input stream should be left in its initial (intact) state when
* calling this function. Upon completion, the next byte read from the stream
* with #s_next() is the first byte of the message-body, if one exists.
*
* In case #HTTPRequest.transfer_encoding is set to #TRANSFER_COD_CHUNK, use of
```

```
* #c_next() gives direct (transparent) access to the entity-body and should be
* preferred over #s_next().
*
* Parsing of the incoming message stops, if the supplied URI has not been
* specified in #server_consts.
*
* @param[in,out] req An #HTTPRequest representation of the HTTP request found on
* the input stream. All @c .query members should be set to appropriate values
* by the time, or when, rsrc_inform() is invoked. The other members will be
* internally initialised to #SRVR_NOT_SET.
*/
void http_parse_request(HTTPRequest* req);

/**
* @brief Extract method, request URI and HTTP version of the request line from
* the stream.
*
* This function is a congregate of the functions stream_match(), parse_uri() and
* parse_http_version().
*
* @param[out] req HTTPRequest variable to be updated with values found on
* stream. Those members are:
* @link HTTPRequest::method method@endlink,
* @link HTTPRequest::uri uri@endlink,
* @link HTTPRequest::v_major v_major@endlink and
* @link HTTPRequest::v_minor v_minor@endlink.
* @param[in,out] c The first character to start parsing from and the last one
* read from the stream.
* @returns One of:
* — CRLF
* — EOF
*/
static int8_t parse_request_line(HTTPRequest* req, uint8_t* c);

/**
* @brief Read the specified transfer-coding from stream.
*
* Only ‘chunked’ and ‘identity’ are currently supported. Should any other, or
* combination thereof, be specified, #TRANSFER_COD_OTHER shall be returned.
*
* As the ‘Transfer-Encoding’ and ‘TE’ headers are a list, this function may be
* invoked more than once for each.
*
* @param[in,out] value One of #TRANSFER_COD_CHUNK, #TRANSFER_COD_IDENT or
* #TRANSFER_COD_OTHER.
* @param[in,out] c The first character to start parsing from and the last one
* read from the stream, typically @c LF.
* @returns One of:
```

```

* — CRLF
* — EOF
*/
static int8_t parse_header_transfer_coding(uint8_t* value, uint8_t* c);

/**
 * @brief Read HTTP major and minor version numbers from stream.
 *
 * @param[out] req HTTPRequest variable to be updated with the message's HTTP
 * version. Members are: @link HTTPRequest::v_major v_major@endlink and
 * @link HTTPRequest::v_minor v_minor@endlink.
 * stream. Which members are actually update depends on what is available.
 * @param[in,out] c The first character to start parsing from and the last one
 * read from the stream.
 * @returns EOF on end of stream; @c 0 or #OTHER, otherwise.
 */
static int8_t parse_http_version(HTTPRequest* req, uint8_t* c);

/**
 * @brief Populates @p req with header values found on the stream.
 *
 * It uses stream_match() to identify headers at the beginning of each line and
 * then
 * forwards the rest of the line to the appropriate header-body handler for that
 * header. Unsupported headers are simply discarded.
 * It terminates on either EOF or a double CRLF (ie, an empty line).
 *
 * @param[out] req HTTPRequest variable to be updated with values found on
 * stream. Which members are actually update depends on what is available.
 * @param[in,out] c The first character to start parsing from and the last one
 * read from the stream.
 * @returns One of:
 * — #CRLF; if an empty line was read. The double CRLF sequence will be dropped
 * from the stream and the last LF returned in @p c.
 * — EOF. Normally, this should not occur in the header section.
 */
static int8_t parse_headers(HTTPRequest* req, uint8_t* c);

/**
 * @brief Parse Accept header body—value is search of media ranges.
 *
 * This function traverses and consumes the stream from its current position up
 * to a CRLF or EOF in an attempt to identify a supported media range (ie, one
 * that is included in the #server_consts).
 *
 * For any identified media range, it also provides its q-value. If more than one
 * q-value is specified for a particular media range, the last one is preserved.
 * For details on the conversion refer to q_value().

```

```
*  
* According to <a href="http://tools.ietf.org/html/rfc2616#section-14.1">IETF  
* RFC 2616 p.100</a>, the Accept header is a list and, so, it may  
* contain more than one media range. The one with the highest q-value is  
* returned.  
*  
* Since the Accept header is a list, it may appear more than once in a request.  
* Should such an occasion arise, the @p media_range and @p qvalue returned by a  
* previous call to this function can be used in its new invocation. This way,  
* they will be taken into consideration while parsing the header-body.  
* Initially, a non-acceptable index of #server_consts and 0, respectively,  
* should suffice.  
*  
* @param[in,out] media_range Index of #server_consts that corresponds to  
* the highest, so far, qvalue-ranking media range. Upon first invocation, it  
* should contain a known invalid value.  
* @param[in,out] qvalue The q-value of the @p media_range. On the first  
* invocation, it should contain a value of 0.  
* @param[out] c The last character read from the stream.  
* @returns One of:  
* - #CRLF  
* - EOF  
*/  
static int parse_header_accept(int8_t* media_range,  
                                uint16_t* qvalue,  
                                uint8_t* c);  
  
/**  
* @brief Match input from stream against the available server endpoints.  
*  
* Absolute paths are recognised as well as absolute URIs; use set_host_name() to  
* provide the server name or IP address. In the case of an absolute URI, the  
* port should match that of the server, if one is specified in the request; see  
* set_host_port().  
*  
* This function is in accordance with  
* <a href="http://tools.ietf.org/html/rfc2616#section-5.1.2">IETF RFC 2616  
* p.37</a> in that if the Request-URI is encoded using the "% HEX HEX" encoding,  
* it decodes the corresponding characters. To do so, it uses stream_match_ext()  
* to be informed of cases that contain a percent sign. If the encoding is valid  
* (ie, a hexadecimal number), the decoded character is fed back into  
* stream_match_ext() and the operation continues as normal.  
*  
* @param[out] req HTTPRequest variable to be updated with the endpoint value  
* found on (@link HTTPRequest::uri uri@endlink).  
* @param[in,out] c The first character to start parsing from and the last one  
* read from the stream.  
* @returns One of:
```

```

* — CRLF
* — EOF
*/
static int8_t parse_uri(HTTPRequest* req, uint8_t* c);

/**
* @brief Read query string parameter values into @p q.
*
* @link QueryString q@endlink should be initialised *before* calling this
* function.
*
* @param[in,out] q An initialised #QueryString variable to accept parameter
* values.
* @param[in,out] c The first character to start discarding from and the last one
* read from the stream. Upon invocation, it should point at '?'.
* @returns One of:
* — @c 0; if a space character has been found. That character is found in @p
* @c .
* — #CRLF; if a premature CRLF sequence has been found.
* — #EOF
*/
static inline int8_t parse_qparams(QueryString* q, uint8_t* c);

/**
* @brief Matches the current name of the server (host) against the stream.
*
* To see how to set the host name of the server, refer to set_host_name().
*
* @param[in,out] c The first character to start comparing from and the last one
* read from the stream.
* @returns One of:
* — @c 0; if @verbatim "http://" "/" host @endverbatim was matched.
* — #OTHER; if either the scheme or the host name failed to match.
* — EOF; if the end of stream was reach in the meantime.
*/
static int8_t parse_host(uint8_t* c);

/**
* @brief Identify and read a q-value parameter.
*
* It should be invoked after an Accept header has been identified on (and
* consumed from) the stream. Any LWS on the stream is discarded without
* affecting its output.
*
* This function is provided on the premise that interest lies in extracting only
* q-value parameters from the stream.
*
* @param qvalue

```

```
* @param c
* @returns One of:
* - #OTHER
* - #CRLF
* - EOF
*/
static int8_t parse_header_param_qvalue(uint16_t* qvalue, uint8_t* c);

/**
* @brief Read a q-value from stream.
*
* According to <a href="http://tools.ietf.org/html/rfc2616#section-3.9">IETF RFC
* 2612 p.29</a>, a quality value is a short "floating point"
* number ranging from 0 up to 1 with, at most, three decimal points. The BNF is:
* @verbatim
qvalue = ( '0' [ '.' 0*3DIGIT ] )
| ( '1' [ '.' 0*3('0') ] )@endverbatim
*
* The q-value is actually read from stream as an integer (per mil).
*
* @param[out] value The q-value read from stream.
* @param[in,out] c The first character to start parsing from and the last one
* read.
* @returns One of:
* - #OTHER
* - EOF
*/
int8_t q_value(uint16_t* value, uint8_t* c);

/**
* @brief Read the next byte from a chunked HTTP message into @p c.
*
* In case the bytes of a particular chunk have been depleted, the first byte of
* the next chunk (if one exists) is returned. This function should only be
* called if a 'Transfer-Encoding' header with a value of '@c chunked' has been
* specified.
*
* @param[out] c The character read from the stream.
* @returns @c 0 on a successful read; @c EOF, otherwise.
*/
int8_t c_next(uint8_t* c);

/**
* @brief Parse the size of the next chunk.
*
* This function is called by front-end functions.
*
* According to <a href="http://tools.ietf.org/html/rfc2616#section-3.6.1">IETC
```

```

* RFC 2616 p.26</a> "All HTTP/1.1 applications MUST be able to receive and
* decode the @c chunked transfer-coding, and MUST ignore chunk extensions they
* do not understand". Only the size is parsed as no chunk extensions are
* currently supported. An extract of the BNF is as follows: @verbatim
Chunked-Body = *chunk
               last-chunk
               trailer
               CRLF

chunk = chunk-size [ chunk-extension ] CRLF
       chunk-data CRLF
chunk-size = 1*HEX
last-chunk = 1*(‘0’) [ chunk-extension ] CRLF

chunk-data = chunk-size(OCTET)@endverbatim
*
* The amount of octets of each chunk is given as a hexadecimal number (excluding
* the CRLF sequence). The last chunk size should evaluate to 0. The trailer is
* also ignored because, as stated by the same source as above, it consists
* entirely of optional metadata that may be discarded before reaching the
* client (in this case, they are discarded by the client).
*
* @param[in,out] c The first character to start parsing from and the last one
* read from the stream.
* @returns @c EOF on end of stream; @c 0, otherwise.
*/
static int8_t update_chunk(uint8_t* c);

/**
* @brief Advances the stream until all successive linear white space has been
* read.
*
* This function is mostly useful for removing spacing as well as header folding
* within a header body. A linear white space -- LWS, for short -- is defined by
* <a href="http://tools.ietf.org/html/rfc822#section-3.3">IETF RFC 288 p.10</a>,
* as: @verbatim 1*([CRLF] LWSP-char) @endverbatim
* ie, an optional CRLF sequence immediately followed by a SPACE or HTAB any
* amount of times. For clarification, a CRLF sequence followed by SPACE or HTAB,
* causes header folding and not its termination.
*
* @param[in,out] c The first character to start discarding from and the last one
* read from the stream.
* @returns One of:
* -- #CRLF; for header termination.
* -- #OTHER; if any non-LWS has been read. That character will have been read
* into @p c.
* -- EOF; if the end of stream has been reached.
*/

```

```
static int8_t discard_LWS(uint8_t* c);

<太后
* @brief Discard everything up to an empty line (ie, two CRLF sequence in
* succession).
*
* @param[in,out] c The first character to start discarding from and the last one
* read from the stream.
* @returns One of:
* - #CRLF; on an empty line.
* - @c EOF; on end-of-stream.
*/
static int8_t discard_to_line(uint8_t* c);

<太后
* @brief Discards the HTTP header-value parameter starting at @p c.
*
* Discards all characters on the stream until a ";" (end of current parameter),
* a "," (end of header-value), a CRLF (end of header) or EOF has occurred.
* In either delimiter ";" or ",", #OTHER is returned and the actual delimiter
* is found in @p c.
*
* Any quoted strings encountered along the way (and quoted-pairs therein) are
* rejected as well.
* Delimiters that occur within the boundaries of quoted strings are handled as
* ordinary text and do not terminate execution. Also, escaped double quotes
* (quoted-pair) are handled accordingly. It should be noted that although a '\"'
* within a quoted string should be able to escape a CR, it is not permitted to
* do so, if that would cancel a CRLF sequence or header folding.
*
* @param[out] c The first character to start discarding from and the last one
* read from the stream.
* @returns One of:
* - #OTHER; if ";" or "," was read.
* - #CRLF
* - EOF
*/
static int8_t discard_param(uint8_t* c);

<太后
* @brief Checks whether there is a LWS on stream starting with @p c.
*
* A linear white space is an LWSP-char (ie, @c SPACE or @c HTAB) optionally
* preceded by a CRLF sequence.
*
* This function does not consume bytes from the stream. It simple peeks forward.
*
* @param c The first character to use in the comparisons.
```

```

* @returns @c 1, if true; @c 0, otherwise.
*/
static int8_t is_LWS(uint8_t c);

/**
* @brief Checks whether there is a CRLF sequence on stream starting with @p c.
*
* It returns @c 0 (ie, false), if there is a CRLF sequence followed by an @c
* LWS—char which, semantically, is a header folding.
*
* This function does not consume bytes from the stream. It simply peeks forward.
*
* @param c The first character to use in the comparisons.
* @returns @c 1, if true; @c 0, otherwise.
*/
static int8_t is_CRLF(uint8_t c);

/**
* @brief Checks whether there is a CRLF sequence on stream starting with @p c.
*
* Unlike is_CRLF(), this function is satisfied with a CRLF sequence no matter
* what follows next, be it a LWS—char or otherwise.
*
* This function does not consume bytes from the stream. It simply peeks forward.
*
* @param c The first character to use in the comparisons.
* @returns @c 1, if true; @c 0, otherwise.
*/
static int8_t is_c_CRLF(uint8_t c);

#endif /* WEB_SERVER_H_INCL */
/** @} */

```

### http\_parser.c

```

#include "http_parser.h"
#include "sbuffer.h"
#include "stream_util.h"

#include <stdio.h>
#include <ctype.h> /* issxdigit(), tolower() */

static ServerSettings* srvr;

/**
* @ingroup http_parser
* @brief Denotes whether a chunked message is already in process.
*/

```

```
static uint8_t is_chunk_on;

/**
 * @ingroup http_parser
 * @brief Amount of total bytes to read from the current chunk.
 */
static uint16_t chunk_len;

/**
 * @ingroup http_parser
 * @brief Amount of bytes read from the current chunk.
 */
static uint16_t chunk_pos;

void http_parser_set_server(ServerSettings* new_settings) {
    srvr = new_settings;
}

void http_parse_request(HTTPRequest* req) {
    uint8_t c;
    int8_t c_type;
    req->method = SRVR_NOT_SET;
    req->uri = SRVR_NOT_SET;
    req->v_major = SRVR_NOT_SET;
    req->v_minor = SRVR_NOT_SET;
    req->accept = SRVR_NOT_SET;
    req->content_type = SRVR_NOT_SET;
    req->content_length = SRVR_NOT_SET;
    req->transfer_encoding = SRVR_NOT_SET;

    /* Parse request- or status-line. */
    c_type = s_next(&c);
    c_type = parse_request_line(req, &c);

    if(req->uri == SRVR_NOT_SET) return;

    /* Parse headers. */
    c_type = s_next(&c); /* Discard LF and load next character. */
    c_type = parse_headers(req, &c);

    /* After parsing headers, if CRLF was returned, an empty line is implied. */

    if(req->transfer_encoding == TRANSFER_COD_CHUNK) {
        req->content_length = 0; /* Ignore Content-Length, if set. */

        /* Since the current request is in chunked coding, it is highly likely
         * that chunk parsing will be necessary. Reset chunk variables to be
         * ready for it. */
    }
}
```

```

        is_chunk_on = 0;
        chunk_len = 0;
        chunk_pos = 0;
    }
}

int8_t parse_request_line(HTTPRequest* req, uint8_t* c) {
    int8_t c_type;

    /* Retrieve the method and discard SP. */
    c_type = stream_match(&srvr->consts[METHOD_MIN], METHOD_MAX, c);
    if(c_type >= 0) req->method = c_type;
    while(*c == ' ') c_type = s_next(c);

    /* Retrieve URI info (host, path, query) */
    c_type = parse_uri(req, c);
    while(*c == ' ') c_type = s_next(c);

    /* Retrieve HTTP version. */
    c_type = parse_http_version(req, c);

    /* Discard the rest of the line. */
    while(!is_CRLF(*c) && c_type != EOF) c_type = s_next(c);
    if(*c == '\r') {
        s_next(c);
        c_type = CRLF;
    }

    return c_type;
}

int8_t parse_http_version(HTTPRequest* req, uint8_t* c) {
    int8_t c_type;
    uint8_t digits;

    c_type = stream_match(&srvr->consts[HTTP_SCHEME], HTTP_SCHEME + 1, c);

    if(c_type >= 0 && *c == '/') {

        s_next(c);
        c_type = parse_uint8(&(req->v_major), c);

        while(*c != '.' && c_type != EOF && !is_CRLF(*c)) {
            c_type = s_next(c);
        }

        if(*c == '.') {
            c_type = s_next(c);
        }
    }
}

```

```

    c_type = parse_uint8(&(req->v_minor), c);
}
}

return c_type;
}

int8_t parse_headers(HTTPRequest* req, uint8_t* c) {
    uint16_t qvalue = 0;
    uint8_t is_emptyln = 0;
    int8_t c_type = 0;
    int8_t idx;
    uint8_t peek;

    while(c_type != EOF && !is_emptyln) {
        /* Attempt to identify a header */
        c_type = stream_match(&srvr->consts[HEADER_MIN], HEADER_MAX, c);

        /* If there is a potential match terminated by a ":", then that match is
         * valid. (No sort spacing is allowed in the header-name.) */
        if(c_type >= 0 && *c == ':') {
            idx = c_type;
            *c = '_'; /* Discarding starts from the provided byte. */
            c_type = discard_LWS(c); /* Ignore LWS on stream. */

            /* Read header-body, if the header-name is not followed by EOF or
             * a CRLF sequence. */
            if(c_type == OTHER) {
                if(idx == HEADER_ACCEPT) {
                    c_type = parse_header_accept(&(req->accept), &qvalue, c);
                } else if(idx == HEADER_CONTENT_LENGTH) {
                    c_type = parse_uint16(&(req->content_length), c);
                } else if(idx == HEADER_TRANSFER_ENC) {
                    c_type = parse_header_transfer_coding(
                        &(req->transfer_encoding), c);
                }
            }
        }

        /* In case of an unsupported header, the line should be discarded. */
    } else {
        do {
            if(*c == '\r' && is_CRLF(*c)) {
                c_type = CRLF;
                s_next(c); /* Load LF into @c c. */
                break;
            }
        } while((c_type = s_next(c)) != EOF);
    }
}

```

```

/* Check whether the end of headers has been reached. */
if(c_type == CRLF) {
    s_peek(&peek, 0);

    if(peek == '\r') {
        s_peek(&peek, 1);

        if(peek == '\n') {
            s_drop(2); /* Discard the second CRLF sequence. */
            is_emptyln = 1;
        }
    }

    /* There exists a new header; read its first byte. */
} else {
    c_type = s_next(c);
}
}

return c_type;
}

int8_t parse_header_transfer_coding(uint8_t* value, uint8_t* c) {
    int8_t c_type;

    /* Combinations of transfer-codings are not supported. If none has been
     * previously specified attempt to identify this one. */
    if(*value == 0) {

        c_type =
            stream_match(&srvr->consts[TRANSFER_COD_MIN], TRANSFER_COD_MAX, c);

        /* An acceptable transfer-coding has been found; pass it into @p req.
         * As a note, punctuation characters (comma, in particular) is used to
         * specify multiple values in a list header. */
        if(c_type >= 0 && !isalpha(*c) && !ispunct(*c) ) {
            *value = c_type;
        } else {
            *value = TRANSFER_COD_OTHER;
        }
    }

    /* Discard the rest of the line. If any combination of transfer-codings is
     * specified, simply fail all values. */
    while(c_type != EOF && !is_CRLF(*c)) {

        if(isalpha(*c)) *value = TRANSFER_COD_OTHER;
        c_type = s_next(c);
    }
}

```

```
    return c_type;
}

int parse_header_accept(int8_t* media_range, uint16_t* qvalue, uint8_t* c) {
    int8_t c_type; /* The character type that is read last (eg. EOF, CRLF). */
    int8_t idx; /* The potentially matched media range. */

    uint16_t qvalue_new = 0;

    do {
        qvalue_new = 1000; /* Reset q-value to 'preferred'. */

        if(*c == ',') {
            c_type = s_next(c);
            c_type = discard_LWS(c);
            if(c_type == CRLF) break;
        }

        /* Identify the first media range. */
        idx = stream_match(&srvr->consts[MIME_MIN], MIME_MAX, c);
        c_type = discard_LWS(c);

        /* In case that a potential match was terminated by a valid delimiter,
         * try to identify its parameters, and in particular, a q-value. */
        if(idx >= 0) {

            /* An acceptable media range without parameters was specified. In
             * case of an ',', more media ranges may follow. */
            if(c_type == CRLF || *c == ',') {
                *qvalue = 1000;
                *media_range = idx;

            } else if(*c == ';') {

                /* Attempt to identify q-value. The last is preserved. */
                do {
                    c_type = parse_header_param_qvalue(&qvalue_new, c);
                } while(*c == ';');

                /* Update media range if it has a higher q-value than the
                 * previous. */
                if(qvalue_new > *qvalue) {
                    *qvalue = qvalue_new;
                    *media_range = idx;
                }
            }
        }
    }
}
```

```

/* If an unsupported media range was supplied (ie, one that completely
 * failed a match --- idx < 0 --- or one, that although produced a match,
 * contained no delimiter), it is with certainty that the rest of the
 * line may be safely discarded. If, along the way, a '",' is read,
 * the process of identifying a new media range is repeated. */
while(c_type != EOF && c_type != CRLF && *c != ',') {
    c_type = discard_param(c);

    /* <discard_param>() returns on either a ';;' (parameter separator)
     * or a ',' (media range separator). If a supported media range was
     * in effect then all its parameters would have been consumed in a
     * previous loop. So this one could only belong to an unsupported
     * media range and must be discarded. */
    if(*c == ',') s_next(c);
}

} while(*c == ',');

return c_type;
}

static int8_t parse_uri(HTTPRequest* req, uint8_t* c) {
    uint8_t min = 0;
    uint8_t max = srvr->rsrc_len;
    uint8_t cmp_idx = 0;
    uint8_t last_it = 255;
    int8_t c_type = 0;

    /* If the request URI begins with the scheme (ie, HTTP), then an absolute
     * URI is provided and it should match against the server (host) name. If
     * not, an appropriate status code should be returned. */
    if(*c != '/' && *c != '*') {
        if(c_type = parse_host(c)) return OTHER;
    }

    /* Parse absolute path. */
    /* Match stream against available absolute paths taking into consideration
     * possible percent-encoding. */
    while(c_type != EOF && min < max) {
        c_type =
            stream_match_ext(srvr->rsrc_tokens, 0, &min, &max, &cmp_idx, c);

        /* If there is a failed match for the second time for a particular
         * iteration (ie, after attempting a percent-decoding, if a percent sign
         * was present), then there can be no match. */
        if(last_it == cmp_idx) {
            break;
        }
    }
}

```

```

}

/* On a mismatch due to '%', retry the comparison, unless already
 * retried. */
if(c_type == OTHER && *c == '%' && last_it != cmp_idx) {
    uint8_t p1, p2;
    uint8_t value;

    last_it = cmp_idx; /* Keep iteration of last percent decoding. */

    /* Decode the next two characters. */
    s_peek(&p1, 0);
    s_peek(&p2, 1);

    if(isxdigit(p1) && isxdigit(p2)) {
        s_drop(2);
        value = p1 <= '9' ? p1 - '0' : tolower(p1) - 'a' + 10;
        value *= 16;
        value += p2 <= '9' ? p2 - '0' : tolower(p2) - 'a' + 10;

        *c = value;
    } else {
        break;
    }

    /* A possible match has been admitted. */
} else if(c_type >= 0) {
    if(*c == '_' || *c == '?') {
        req->uri = c_type;
    }
    break;
}

/* Certain mismatch. */
} else {
    break;
}
}

/* Initialise query tokens, if a known URI has been found. */
if(req->uri != SRVR_NOT_SET) {

    rsrc_inform(req);

    /* Parse query parameters, if a question mark has been found and
     * parameters are applicable for this URI and method. */
    if(*c == '?' && req->query.count) {
        c_type = parse_qparams(&req->query, c);
    }
}

```

```

}

/* Discard while space. */
while(c_type != EOF && *c == ' ') {
    c_type = s_next(c);
}

return c_type;
}

static inline int8_t parse_qparams(QueryString* q, uint8_t* c) {
    int8_t c_type;
    uint8_t param_i; /* Index of an identified parameter. */
    uint16_t i; /* Offset in query.buf[] to write to next. */

    c_type = s_next(c); /* Read next of '?'. */

    /* Repeat while there are characters in the query of the request line. */
    while(c_type != EOF && !is_CRLF(*c) && *c != ' ') {

        /* Attempt to find a parameter token. */
        c_type = stream_match(q->tokens, q->count, c);

        /* A parameter token is found, if the delimiter is the 'equals' sign,
         * after which a value might follow. Accept its value, if one has not
         * been previously specified. */
        if(c_type >= 0 && *c == '=' && q->values[c_type] == NULL) {

            param_i = c_type; /* Store parameter index. */
            i = q->buf_i; /* Offset in query.buf[] . */

            c_type = s_next(c); /* Read next of '='. */

            /* Copy parameter value up to an EOF, space, ampersand or the end
             * of q->buf, whichever happens first. */
            while(c_type != EOF && !is_CRLF(*c)
                  && *c != ' ' && *c != '=' && i < q->buf_len - 1) {

                q->buf[i++] = *c; /* Copy character. */
                c_type = s_next(c); /* Load next character. */
            }

            /* Specify that the parameter buffer has been exceeded. */
            if(i == q->buf_len - 1 && !is_CRLF(*c)) {

                /* TODO: Better way to show the URI is too long. */
                i = q->buf_len;
                q->buf_i = i; /* Buffer is full. */
            }
        }
    }
}

```

```
/* Parameter was found with at least one character in its value. */
} else if(i != q->buf_i) {

    /* Store the start address of the value-string. */
    q->values[param_i] = &q->buf[q->buf_i];

    /* Append a null-byte. */
    q->buf[i++] = '\0';

    /* Set the offset for the next string. */
    q->buf_i = i;
}

/* Not an acceptable parameter; discard it. */
} else {

    while(c_type != EOF && !is_CRLF(*c) && *c != '&' && *c != '_') {
        c_type = s_next(c);
    }
    if(*c == '&') c_type = s_next(c);
}

if(is_CRLF(*c)) c_type = CRLF;
return c_type;
}

static int8_t parse_host(uint8_t* c) {
    int8_t c_type;
    uint8_t* ptr[1];

    /* Match scheme 'HTTP://' */
    c_type = stream_match(&srvr->consts[HTTP_SCHEME_S], HTTP_SCHEME_S + 1, c);

    /* If scheme is acceptable, parse the host name. */
    if(c_type >= 0) {
        ptr[0] = srvr->host_name;
        c_type = stream_match(ptr, 1, c);

        /* Should the host name also be acceptable, parse the port, if one is
         * specified. */
        if(c_type >= 0) {

            /* If a port was provided, ensure it matches the host's. */
            if(*c == ':') {
                c_type = s_next(c); /* Get character next to colon. */
                ptr[0] = srvr->host_port;
            }
        }
    }
}
```

```

c_type = stream_match(ptr, 1, c);

if(c_type >= 0) return 0;

/* An unsupported port was specified; discard it. Odd if this
 * happened but still needs to be dealt with. */
else {
    while(isdigit(*c)) c_type = s_next(c);
}
} else {
    return 0;
}
}

return c_type;
}

static int8_t parse_header_param_qvalue(uint16_t* qvalue, uint8_t* c) {
int8_t c_type;

c_type = s_next(c);

if(c_type != EOF) {
    c_type = discard_LWS(c); /* Ignore any linear white space. */

/* Check whether the function above was terminated because a q-value
 * parameter has occurred. */
if(*c == 'q') {
    *c = '_';
    c_type = discard_LWS(c);

    if(*c == '=') {
        *c = '_';
        c_type = discard_LWS(c);

        /* Try to convert q-value from stream. If it is not a valid
         * value, call discard_param() on the rest of the input. */
        if(c_type == OTHER) {
            q_value(qvalue, c);
        }
    }
}

if(*c != ';' && *c != ',') {

/* Discard the (rest of the) parameter. */
c_type = discard_param(c);
}
}

```

```
    }

    return c_type;
}

int8_t q_value(uint16_t* value, uint8_t* c) {
    uint8_t i;
    int8_t c_type = OTHER;

    *value = 0;

    if(*c == '0') {
        c_type = s_next(c);

        if(*c == '.') {
            /* A maximum of three digits are read; the rest are dropped. */
            c_type = s_next(c);
            for(i = 0 ; i < 3 && *c >= '0' && *c <= '9' ; ++i) {
                *value *= 10;
                *value += *c - 0x30;
                c_type = s_next(c);
            }
        }

        /* Imitate 3 digits had been given. */
        for(i ; i < 3 ; ++i) {
            *value *= 10;
        }
    }
} else {
    *value = 1000;
}
return c_type;
}

int8_t c_next(uint8_t* c) {
    if(chunk_pos == chunk_len) {
        update_chunk(c);
    }

    if(chunk_pos < chunk_len) {
        ++chunk_pos;
        return s_next(c);
    }
    is_chunk_on = 0;
    chunk_len = 0;
    chunk_pos = 0;
    return EOF;
}
```

```

static int8_t update_chunk(uint8_t* c) {
    int8_t c_type = 0;

    if(!is_chunk_on) {
        is_chunk_on = 1;

        /* Discard */
        while(!isxdigit(*c) && c_type != EOF) c_type = s_next(c);

    } else {
        /* Discard CRLF to advance to the size of the next chunk. Folding has no
         * hold in this context, so disregard it. */
        while(!is_c_CRLF(*c) && (c_type = s_next(c)) != EOF);
        if(is_c_CRLF(*c)) {
            s_drop(1); /* Drop LF. */
            c_type = s_next(c); /* Read first digit. */
        }
    }

    chunk_pos = 0;
    chunk_len = 0;
    c_type = parse_hex16(&chunk_len, c);

    /* Discard the rest of the line. */
    while(c_type != EOF && !is_c_CRLF(*c)) c_type = s_next(c);
    if(is_c_CRLF(*c)) {
        c_type = s_next(c); /* Load LF and advance pointer to the first byte. */
    }

    /* If the last chunk was read, discard trailer headers (if any) up to, and
     * including, the final empty line. */
    if(chunk_len == 0) {
        /* @c c would either be a CR (of the terminating CRLF sequence) or the
         * start of a new header. */
        c_type = discard_to_line(c);
    }

    return c_type;
}

static int8_t discard_LWS(uint8_t* c) {
    uint8_t peek;

    do {
        if(*c == '\r') {
            if(s_peek(&peek, 0)) return EOF;

            if(peek == '\n') {

```

```

    if(s_peek(&peek, 1)) return EOF;

    /* A CR followed by a LN and a LWSP-char is a valid LWS. */
    if(peek == '\u' || peek == '\t') {
        /* Discard LF and LWSP-char from the stream. */
        s_drop(2);
        *c = peek;
        continue;
    }
    *c = '\n';
    s_drop(1); /* Discard LF from stream. */
    return CRLF;
}

return OTHER;

/* A single LWSP-char is a valid LWS, anything else is not. */
} else if (*c != '\u' && *c != '\t') {
    return OTHER;
}
} while(!s_next(c));

return EOF;
}

static int8_t discard_to_line(uint8_t* c) {
    int8_t c_type = 0;
    uint8_t is_emptyln = 0;
    uint8_t peek;

    while(!is_emptyln && c_type != EOF) {
        /* Discard bytes until a CRLF. */
        while((c_type = s_next(c)) != EOF && !is_CRLF(*c));

        /* Should a CRLF be followed by another CRLF, then an empty line has
         * been reached. */
        if(is_CRLF(*c)) {
            s_peek(&peek, 1); /* If an empty line, this should be a CR. */

            if(peek == '\r') {
                s_peek(&peek, 2);

                /* A second CRLF has been spotted. */
                if(peek == '\n') {
                    s_drop(3); /* Discard the second LF/CRLF sequence. */
                    is_emptyln = 1;
                    c_type = CRLF;
                }
            }
        }
    }
}

```

```

        }
    }

    return c_type;
}

static int8_t discard_param(uint8_t* c) {
    /* Identifies whether a quoted-string is currently traversed. */
    uint8_t is_quoted_string = 0;

    /* The return status of this function. Generally, it describes the type of
     * the last character read. */
    int8_t c_type = 0;
    uint8_t peek;

    /* While none of CRLF, EOF or OTHER has been found, proceed with the next
     * character. */
    do {
        /* It should be noted that the stream may contain double quotes,
         * in which case they should be matched to an appropriate matching quote
         * before identifying a ',"' or a ',"'. Also, within quoted-strings, a '\'
         * creates a quoted-pair, effectively escaping the following character.
         * For example, if that character is a double-quote, it should not
         * terminate the quoted-string. */

        /* Although a '\\" should be able to escape a CR, it is not permitted to
         * do so, if that would cancel a CRLF sequence or folding. */

        if(*c == '\\" && is_quoted_string) {
            /* The following code block is equivalent to:
             * if(!is_CRLF(*c) || !is_LWS(*c) && *c != ' ' && *c != '\t') {
             *     s_drop(1);
             * }
             */
            c_type = s_peek(&peek, 0);

            if(peek != '\r') {
                *c = peek;
                s_drop(1);
            } else {
                c_type = s_peek(&peek, 1);

                if(*c != '\n') {
                    *c = peek;
                    s_drop(1);
                }
            }
        }
    }

    /* A quoted-string is initiated and terminated by a double quote. */
}

```

```

} else if(*c == '') {
    is_quoted_string = !is_quoted_string;

} else if(*c == '\r') {
    /* Identify CRLF and header folding. */
    c_type = s_peek(&peek, 0);

    if(peek == '\n') {
        c_type = s_peek(&peek, 1);

        if(peek == '\u' || peek == '\t') {
            *c = peek; /* Update @c c with the character read. */
            s_drop(2); /* Discard folding from the stream. */
        } else {
            *c = '\n';
            c_type = CRLF;
            s_drop(1);
            break;
        }
    }
}

} else if(!is_quoted_string && (*c == ';' || *c == ',')) {
    c_type = OTHER;
    break;
}

} while((c_type = s_next(c)) == 0);

return c_type;
}

int8_t is_LWS(uint8_t c) {

    if(c == '\u' || c == '\t') return 1;

    if(c == '\r') {
        if(s_peek(&c, 0)) return 0;

        if(c == '\n') {
            if(s_peek(&c, 1)) return 0;

            /* A CR followed by a LN and a LWSP-char is a valid LWS. */
            if(c == '\u' || c == '\t') {
                return 1;
            }
        }
    }
    return 0;
}

```

```

int8_t is_CRLF(uint8_t c) {

    if(c == '\r') {
        if(s_peek(&c, 0)) return 0;

        if(c == '\n') {
            if(s_peek(&c, 1)) return 0;

            /* A CRLF is rendered a LWS, if it is followed by a LWS-char, ie,
             * SPACE or HTAB. */
            if(c != ' ' && c != '\t') return 1;
        }
    }

    return 0;
}

int8_t is_c_CRLF(uint8_t c) {

    if(c == '\r') {
        if(s_peek(&c, 0)) return 0;

        if(c == '\n') return 1;
    }

    return 0;
}

```

## Κορυός διαχομιστή HTTP

### http\_server.h

```

/**
 * @file
 * @addtogroup http_server HTTP Server
 * @{
 */

#ifndef HTTP_SERVER_H_INCL
#define HTTP_SERVER_H_INCL

#include "resource.h"

#include <inttypes.h>

/**
 * @brief Maximum server name length (including null-byte).
 */
#define HOST_NAME_LEN 16

```

```
/**  
 * @brief Maximum server port length (including null-byte).  
 */  
#define HOST_PORT_LEN 6  
  
/**  
 * @brief Size of the longest TXF_* macro.  
 *  
 * To conserve SRAM, some text fragments (TXF_* macros) are stored in program  
 * space (Flash memory) and only loaded into main memory when required. This  
 * macro defines the size of the automatic (local) buffer that will be populated  
 * with any requested text fragments and should be large enough to contain the  
 * longest TXF_* string.  
 */  
#define TXF_BUF_LEN 44  
  
#ifndef NULL  
/**  
 * Specify that a pointer has not been set to a valid address.  
 */  
#define NULL 0  
#endif  
  
/**  
 * @brief Various HTTP server settings.  
 *  
 * The settings include the Host name and server port,  
 */  
typedef struct {  
  
    /**  
     * @brief A list of tokens used in parsing HTTP headers.  
     *  
     * For a detailed description, see #server_consts.  
     */  
    uint8_t** consts;  
  
    /**  
     * @brief The name of the server (value of 'Host' header and part of absolute  
     * URIs).  
     *  
     * This value is compared against (and must match) the host of the request  
     * line of incoming HTTP requests that specify an absolute URI. It is set  
     * using srvr_set_host_name() or srvr_set_host_name_ip(). It should be noted  
     * that no trailing slash should ever be appended.  
     */  
    uint8_t host_name[HOST_NAME_LEN];
```

```

/**
 * @brief The listening port of the server; defaults to 80.
 *
 * This value is compared against (and must match) the port of the request
 * line of incoming HTTP requests that specify one. It is set using
 * srvr_set_port().
 */
uint8_t host_port[HOST_PORT_LEN];

/**
 * @brief Reference to the supported absolute path tokens.
 *
 * The value is provided by the resource.h module during its initialisation
 * by calling srvr_set_resource().
 */
uint8_t** rsrc_tokens;

/**
 * @brief An array of resource handlers one for each token in @c rsrc_tokens.
 *
 * Each handler in this array is invoked when the token in the corresponding
 * position of @c rsrc_tokens is found in the HTTP request URI, granted there
 * is a #MethodFlag bit set for that request method (for details, see
 * #ResourceHandler).
 *
 * The value is provided by the resource.h module during its initialisation
 * by calling srvr_set_resource().
 */
ResourceHandler* rsrc_handlers;

/**
 * @brief The number of token-handler pairs found in @c rsrc_tokens and @c
 * rsrc_handlers.
 *
 * The value is provided by the resource.h module during its initialisation
 * by calling srvr_set_resource().
 */
uint8_t rsrc_len;
} ServerSettings;

/**
 * @brief A representation of an HTTP message.
 */
typedef struct HTTPRequest {
    /** @brief Value representing the method of the request. */
    uint8_t method;
}

```

```
/** @brief Value representing the URI of the request line. */
uint8_t uri;

/** @brief The major number of the HTTP version of the message. */
uint8_t v_major;

/** @brief The minor number of the HTTP version of the message. */
uint8_t v_minor;

/** @brief Value representing the accept media range of the request. */
int8_t accept;

/** @brief Value representing the transfer encoding of the message. */
uint8_t transfer_encoding;

/** @brief Value representing the content type of the message. */
uint8_t content_type;

/** @brief The length (in octets) of the message. */
uint16_t content_length;

/** 
 * @brief Permissible query parameter tokens.
 *
 * Passing an #HTTPRequest variable to rsrc_inform(), *after* .method and
 * .uri have been set, will update
 * @link QueryString#tokens .tokens@endlink and
 * @link QueryString#count .count@endlink of .query to the acceptable options
 * for that Resource and method.
 */
QueryString query;
} HTTPRequest;

/** 
 * @brief The total amount of text fragments that may be used with
 * srvr_compile().
 */
#define TXF_MAX 27
#define TXF_SPACE 0 /**< @brief A single space. */
#define TXF_COLON 1 /**< @brief A single colon. */
#define TXF_CRLF 2 /**< @brief A CRLF sequence (0x0D, 0x0A). */
#define TXF_STATUS_200 3 /**< @brief The text: 200 OK */
#define TXF_STATUS_404 4 /**< @brief The text: 404 Not Found */
#define TXF_STATUS_405 5 /**< @brief The text: 405 Method Not Allowed */
#define TXF_STATUS_501 6 /**< @brief The text: 501 Not Implemented */
#define TXF_HTTPv 7 /**< @brief The text: HTTP/1.1 */
#define TXF_ALLOW 8 /**< @brief The text: Allow */
#define TXF_CONNECTION_CLOSE 9 /**< @brief The text: Connection: close */
```

```

#define TXF_CONTENT_LENGTH 10 /**< @brief The text: Content-Length */
#define TXF_CONTENT_TYPE 11 /**< @brief The text: Content-Type */
#define TXF_SERVER 12 /**< @brief Header Server and its value. */
#define TXF_COMMA 13 /**< @brief A single comma. */
#define TXF_RETRY_AFTER 14 /**< @brief The text: Retry-After */
#define TXF_STATUS_202 15 /**< @brief The text: 202 Accepted */
#define TXF_STATUS_400 16 /**< @brief The text: 400 Bad Request */
#define TXF_STATUS_503 17 /**< @brief The text: 503 Service Unavailable */
#define TXF_SEMICOLON 18 /**< @brief A single semicolon. */
#define TXF_CHUNKED 19 /**< @brief Chunked transfer encoding header. */
#define TXF_CHAR_UTF8 20 /**< @brief The text: charset=utf-8 */
#define TXF_JSON_LINE 21 /**< @brief A complete JSON type header line. */
#define TXF_GZIP_LINE 22 /**< @brief A complete gzip encoding line. */
#define TXF_JS_LINE 23 /**< @brief A complete JS type header line. */
#define TXF_CSS_LINE 24 /**< @brief A complete CSS type header line. */
#define TXF_CACHE_NO_CACHE 25 /**< @brief The text: Cache-Control:no-cache */
#define TXF_CACHE_PUBLIC 26 /**< @brief The text: Cache-Control:public */

/**
 * @brief Alias of #TXF_SPACE.
 */
#define TXF_SP TXF_SPACE

/**
 * @brief Alias of #TXF_COLON.
 */
#define TXF_HS TXF_COLON

/**
 * @brief Alias of #TXF_CRLF
 */
#define TXF_ln TXF_CRLF

/**
 * @brief Empty line (CRLF,CRLF).
 */
#define TXF_lnl TXF_CRLF, TXF_CRLF

/**
 * @brief 'Content-Length' header with a value of @c 0.
 */
#define TXF_CONTENT_LENGTH_ZERO_ln \
TXF_CONTENT_LENGTH, TXF_HS, TXFx_FW_UINT, 0, TXF_ln

/**
 * @brief A set of headers that should be present in all responses.
 */
#define TXF_STANDARD_HEADERS_ln \

```

```
TXF_SERVER, TXF_CRLF, \
TXF_CONNECTION_CLOSE, TXF_CRLF

/***
* @brief Equivalent to #srvr_compile(1, ..., #SRVR_NOT_SET).
*
* Convenience macro to avoid specifying #SRVR_NOT_SET as the last argument and
* @c 1 as the first.
*/
#define srvr_send(...) \
srvr_compile(1, __VA_ARGS__, SRVR_NOT_SET)

/***
* @brief Equivalent to #srvr_compile(0, ..., #SRVR_NOT_SET).
*
* Convenience macro to avoid specifying #SRVR_NOT_SET as the last argument and
* @c 0 as the first.
*/
#define srvr_prep(...) \
srvr_compile(0, __VA_ARGS__, SRVR_NOT_SET)

/***
* @brief Pass any custom string into srvr_compile().
*
* Causes the next argument to be interpreted as a (null-terminated) string which
* is copied into the output buffer as-is.
*/
#define TXFx_FW_STRING 254

/***
* @brief Pass an unsigned number into srvr_compile().
*
* Causes the next argument to be converted into a string and then copied into
* the output buffer.
*/
#define TXFx_FW_UINT 253

/***
* @brief Make the next string printed by srvr_compile() appear in upper-case.
*
* Causes the next text fragment to be converted into upper-case before copying
* it into the output buffer. Note that if a custom string was supplied (with
* #TxFx_TO_ALLCAP), the original string will be altered.
*/
#define TXFx_TO_ALLCAP 252

/***
* @brief Use a text fragment that resides in main memory in srvr_compile().
*/
```

```

*
* Causes the next argument to be interpreted as a constant from #server_consts.
* Currently, they have to be explicitly declared
*/
#define TXFx_FROMRAM 251

/***
* @brief General-context macro for any parameter not set to a known value.
*/
#define SRVR_NOT_SET (0xFF)

/***
* @brief The starting index in #server_consts of supported method literals.
*/
#define METHOD_MIN 0
#define METHOD_CONNECT 0 /*< @brief Method @c CONNECT. */
#define METHOD_DELETE 1 /*< @brief Method @c DELETE. */
#define METHOD_GET 2 /*< @brief Method @c GET. */
#define METHOD_HEAD 3 /*< @brief Method @c HEAD. */
#define METHOD_OPTIONS 4 /*< @brief Method @c OPTIONS. */
#define METHOD_POST 5 /*< @brief Method @c POST. */
#define METHOD_PUT 6 /*< @brief Method @c PUT. */
#define METHOD_TRACE 7 /*< @brief Method @c TRACE. */
/***
* @brief The number of HTTP method tokens.
*/
#define METHOD_MAX 8

/***
* @brief The starting index in #server_consts of supported header literals.
*/
#define HEADER_MIN (METHOD_MAX)
#define HEADER_ACCEPT 0 /*< @brief Header @c Accept. */
#define HEADER_CONTENT_LENGTH 1 /*< @brief Header @c Content-Length. */
#define HEADER_CONTENT_TYPE 2 /*< @brief Header @c Content-Type. */
#define HEADER_TRANSFER_ENC 3 /*< @brief Header @c Transfer-Encoding. */
/***
* @brief The number of HTTP header tokens.
*/
#define HEADER_MAX 4

/***
* @brief The starting index in #server_consts of supported media range literals.
*/
#define MIME_MIN (METHOD_MAX+HEADER_MAX)
#define MIME_ANY 0 /*< @brief Media range ''* / *''. */
#define MIME_APP_ANY 1 /*< @brief Media range ''application/any''. */
#define MIME_APP_JSON 2 /*< @brief Media range ''application/json''. */

```

```
#define MIME_TEXT_ANY 3 /**< @brief Media range "text/" */
#define MIME_TEXT_HTML 4 /**< @brief Media range "text/html". */
#define MIME_TEXT_JSON 5 /**< @brief Media range "text/json". */
/** 
 * @brief The number of media range literals.
 */
#define MIME_MAX 6

/** 
 * @brief The starting index in #server_consts of available transfer-codings.
 */
#define TRANSFER_COD_MIN (METHOD_MAX+HEADER_MAX+MIME_MAX)
#define TRANSFER_COD_CHUNK 0 /**< @brief Chunked transfer-coding. */
#define TRANSFER_COD_IDENT 1 /**< @brief Identity transfer-coding. */
/** 
 * @brief The number of transfer-coding literals.
 */
#define TRANSFER_COD_MAX 2

/** 
 * @brief Describes an unsupported transfer-coding value or combination thereof.
 *
 * This is not a #server_consts index.
 */
#define TRANSFER_COD_OTHER TRANSFER_COD_MAX

/** @brief HTTP literal. */
#define HTTP_SCHEME (METHOD_MAX+HEADER_MAX+MIME_MAX+TRANSFER_COD_MAX)

/** @brief HTTP scheme with separator. */
#define HTTP_SCHEME_S (HTTP_SCHEME + 1)

/** 
 * @brief Convert a METHOD_ macro to a MethodFlag bit.
 */
#define TO_METHOD_FLAG(x) (1<<(x - METHOD_MIN))

/** 
 * @brief Convenience macro to print: Content-Type:application/json;charset=utf-8
 */
#define TXF_CONTENT_TYPE_JSON_ln TXF_JSON_LINE, TXF_ln

/** 
 * @brief Convenience macro to print: Content-Encoding:gzip
 */
#define TXF_GZIP_ln TXF_GZIP_LINE, TXF_ln

/**
```

```

* @brief Convenience macro to print: Content-Type:application/javascript; charset=utf-8
*/
#define TXF_CONTENT_TYPE_JS_ln TXF_CONTENT_TYPE, TXF_HS, \
                            TXF_JS_LINE, TXF_ln

/***
* @brief Convenience macro to print: Content-Type:text/css
*/
#define TXF_CONTENT_TYPE_CSS_ln TXF_CONTENT_TYPE, TXF_HS, \
                            TXF_CSS_LINE, TXF_ln

/***
* @brief Convenience macro to print: Cache-Control:no-cache
*/
#define TXF_CACHE_NO_CACHE_ln TXF_CACHE_NO_CACHE, TXF_ln

/***
* @brief Convenience macro to print: Cache-Control:public
*/
#define TXF_CACHE_PUBLIC_ln TXF_CACHE_PUBLIC, TXF_ln

/***
* @brief HTTP method flag—bits that may be OR-ed together.
*
* There is direct correlation between these bit—flags and METHOD_* macro
* definitions (such as #METHOD_GET and #METHOD_PUT) the latter being used, most
* notably, in #HTTPRequest::method, although they are not the same. Each
* bit—flag is a power of 2 so they may be intermixed to describe any desirable
* combination of methods. On the other hand, the macros are indices where a
* #server_consts HTTP method literal resides in.
*
* Granted that all method literals in #server_consts are stored in succession,
* it is easy to determine its corresponding #MethodFlag, as follows:
* @verbatim
MethodFlag_bit = 1 << (METHOD_* - METHOD_MIN)@endverbatim
* Where:
* — @c METHOD_* is a method macro (or, in other word, a #server_consts index),
* except for #METHOD_MIN.
* — @c METHOD_MIN is the macro for the index of the first method literal.
* — @c MethodFlag_bit the #MethodFlag value, METHOD_* corresponds to.
*
* Also, see #TO_METHOD_FLAG.
*/
typedef enum {
    /** @brief Method OPTIONS flag-bit. */
    HTTP_OPTIONS = TO_METHOD_FLAG(METHOD_OPTIONS),

    /** @brief Method GET flag-bit. */
    HTTP_GET = TO_METHOD_FLAG(METHOD_GET),
}

```

```
/** @brief Method PUT flag-bit. */
HTTP_PUT = TO_METHOD_FLAG(METHOD_PUT),

/** @brief Method POST flag-bit. */
HTTP_POST = TO_METHOD_FLAG(METHOD_POST)

} MethodFlag;

/** 
 * @brief Initialise HTTP server modules.
 *
 * It is enough to call this only once.
 */
void srvr_init();

/** 
 * @brief Register the specified resource tokens and handlers with the server.
 *
 * All paths supported by the server must be issued exactly one handler this way.
 * Failing to do so for some paths will result in a 404 (Not Found) response
 * being returned to the requester entity for those paths. This function need
 * only be invoked once, unless the address of the arrays changes (ie, switching
 * individual handlers during run-time --- via rsrc_set_handler() --- does not
 * affect the array address).
 *
 * Passing @c NULL or @c 0 to any of the parameters discards all resource
 * references.
 *
 * @param[in] tokens Supported absolute path tokens.
 * @param[in] handlers Array of functions to call for each string in @p tokens.
 * @param[in] len The number of token-handler pairs found in @p tokens and @p
 * handlers.
 */
void srvr_set_resources(uint8_t** tokens,
                       struct ResourceHandler* handlers,
                       uint8_t len);

/** 
 * @brief Convert and set an IP address as the host name of the HTTP server.
 *
 * This function is an alternative to set_host_name(). It uses inet_to_str() for
 * the conversion.
 *
 * @param[in] ip A four-byte array of an IP address.
 */
void srvr_set_host_name_ip(uint8_t* ip);

/**
```

```

* @brief Send the chunk size of a chunk.
*
* Insignificant leading zeros are not prohibited by the specification
* (<a href="http://tools.ietf.org/html/rfc2616#section-3.6.1">RFC2616 –
* 3.6.1 Chunked Transfer Coding</a>) and, so, four hexadecimal digits are always
* printed, followed by a @c CRLF sequence. The output is not flushed.
*
* @param[in] len The size of the chunk. This will be converted into hexadecimal
* notation.
* @returns The outcome of send() (for details, see the return value of
* srvr_compile()).
*/
int16_t srvr_prep_chunk_head(uint16_t num);

/**
* @brief Compiles a response based on the specified text fragments.
*
* The header section of all HTTP responses should be compiled from pieces of
* text taken from a common pool of fragments and put together in the appropriate
* order. This is done through this function to avoid re-definition of similar
* fragments and blocks of code, as well as to hide any internal caching
* mechanisms applied on them.
*
* Internally, this function uses send() to communicate the corresponding text
* fragments to the network module which, in turn, preserves them until send() is
* called with its @c flush argument set to a non-zero value. This could be done
* through this function when @p flush holds a non-zero value or directly with
* send(). Additional data may be sent directly to the network module (with
* send()) at any time, regardless whether this function has previously been or
* will be called again.
*
* As stated in the prototype, this function receives any amount of optional
* arguments which correspond to the text fragments to write to the network
* module. For a list of available fragments, see the TXF_* macros specified
* herein. Obviously, the order in which the macros are specified is important
* because it affects the order in which the corresponding text fragments are
* written to the network module. The arbitrary list of optional arguments is
* terminated by passing #SRVR_NOT_SET as the *last argument*. Failing to do so
* involves the risk of stack overflow and arbitrary text fragments sent to the
* network module.
*
* @param[in] flush Designates whether send() should be called with the intention
* to return its buffered data to the requester entity, after the specified
* fragments of this function have been sent to it.
* @param[in] ... Any series of TXF_* macros that specify which text fragments
* and in what order should be sent to the network module. The last argument
* *should always* be #SRVR_NOT_SET.
* @returns The outcome of send(): a non-negative number is the amount of

```

```
* available bytes in the networks module's output buffer after appending the
* specified fragments; a negative number indicates that appending the
* specified fragments has stopped because one of them could not fit in the
* network module's currently available buffer space due to lack of as many
* bytes as the returned value.
*/
int srvr_compile(uint8_t flush, ...);

/**
 * @brief Notify data have arrived on the HTTP server's socket.
 *
 * This function is responsible for performing all necessary initialisation,
 * parsing the incoming data as an HTTP request and returning an appropriate
 * response to the requester entity, either via the use of a user-defined handler
 * (see, #ResourceHandler) or one of predefined messages in case of an exception.
 */
void srvr_call();

#endif /* HTTP_SERVER_H_INCL */
/** @} */
```

### http\_server.c

```
#include "http_server.h"
#include "http_parser.h"
#include "json_parser.h"
#include "resource.h"
#include "w5100.h"
#include "sbuffer.h"
#include "util.h"

#include <avr/pgmspace.h>
#include <stdarg.h>

#include <string.h>

uint8_t txf_space[] PROGMEM = "\0";
uint8_t txf_colon[] PROGMEM = ":";;
uint8_t txf_CRLF[] PROGMEM = "\r\n";
uint8_t txf_status_200[] PROGMEM = "200\OK";
uint8_t txf_status_202[] PROGMEM = "202\Accepted";
uint8_t txf_status_400[] PROGMEM = "400\Bad\Request";
uint8_t txf_status_404[] PROGMEM = "404\Not\Found";
uint8_t txf_status_405[] PROGMEM = "405\Method\Not\Allowed";
uint8_t txf_status_501[] PROGMEM = "501\Not\Implemented";
uint8_t txf_status_503[] PROGMEM = "503\Service\Unavailable";
uint8_t txf_HTTPv[] PROGMEM = "HTTP/1.1";
uint8_t txf_allow[] PROGMEM = "Allow";
uint8_t txf_connection_close[] PROGMEM
```

```

        = "Connection:close";
uint8_t txf_content_length[] PROGMEM
        = "Content-Length";
uint8_t txf_content_type[] PROGMEM = "Content-Type";
uint8_t txf_retry_after[] PROGMEM = "Retry-After";
uint8_t txf_server[] PROGMEM = "Server:uServer\u2192(TEIA)";
uint8_t txf_comma[] PROGMEM = ",";
uint8_t txf_semicolon[] PROGMEM = ";";
uint8_t txf_chunked[] PROGMEM = "Transfer-Encoding:chunked";
uint8_t txf_char_utf8[] PROGMEM = "charset=utf-8";
uint8_t txf_JSON_line[] PROGMEM
        = "Content-Type:application/json; charset=utf-8";
uint8_t txf_gzip_line[] PROGMEM = "Content-Encoding:gzip";
uint8_t txf_JS_line[] PROGMEM = "text/javascript; charset=utf-8";
uint8_t txf_css_line[] PROGMEM = "text/css";
uint8_t txf_cache_no[] PROGMEM = "Cache-Control:no-cache";
uint8_t txf_cache_public[] PROGMEM = "Cache-Control:public";

/* Doxygen does not handle attributes (like PROGMEM) very well. */
/*
 * @ingroup http_server
 * @brief Text fragments stored in program space (Flash memory).
 *
 * In order to conserve main memory (SRAM), some text fragments are stored into
 * the more abundant program memory and loaded into main memory, as needed.
 */
PGM_P srvr_txf[] PROGMEM = {
    txf_space,
    txf_colon,
    txf_CRLF,
    txf_status_200,
    txf_status_404,
    txf_status_405,
    txf_status_501,
    txf_HTTPv,
    txf_allow,
    txf_connection_close,
    txf_content_length,
    txf_content_type,
    txf_server,
    txf_comma,
    txf_retry_after,
    txf_status_202,
    txf_status_400,
    txf_status_503,
    txf_semicolon,
    txf_chunked,
}

```

```
txf_char_utf8,
txf_JSON_line,
txf_gzip_line,
txf_JS_line,
txf_css_line,
txf_cache_no,
txf_cache_public
};


```

```

'put',
'trace',
/* HEADERS, min: METHODS, max: 4 */
'accept',
'content-length',
'content-type',
'transfer-encoding',
/* MEDIA RANGES, min: METHODS+HEADERS, max: 6 */
'*/*',
'application/*',
'application/json',
'text/*',
'text/html',
'text/json',
/* TRANSFER_CODING, min: METHODS+HEADERS+MEDIA_RANGES, max: 2 */
'chunked',
'identity',
/* HTTP TOKENS, indices: METHODS+HEADERS+MEDIA_RANGES+T_CODING, +1 */
'http',
'http://'
};

static ServerSettings srvr = {
    .consts = server_consts,

    .host_name = "000.000.000.000",
    .host_port = "80",

    .rsrc_tokens = NULL,
    .rsrc_handlers = NULL,
    .rsrc_len = 0
};

void srvr_init() {
    /* Provide a reference of server settings to the HTTP parser. */
    http_parser_set_server(&srvr);

    /* Specify which parser should be used by the resource handlers. Generally,
     * this should be done every time before calling a resource handler. But, in
     * the current implementation, only JSON formatted data are supported. */
    rsrc_set_parser(&json_parse);
    rsrc_set_serial(&json_serialise);
}

void srvr_set_resources(uint8_t** tokens,
                       ResourceHandler* handlers,
                       uint8_t len) {

```

```

if(tokens && handlers && len) {
    srvr.rsrc_tokens = tokens;
    srvr.rsrc_handlers = handlers;
    srvr.rsrc_len = len;
} else {
    srvr.rsrc_tokens = NULL;
    srvr.rsrc_handlers = NULL;
    srvr.rsrc_len = 0;
}
}

void srvr_set_host_name_ip(uint8_t* ip) {
    inet_to_str(srvr.host_name, ip);
}

int16_t srvr_prep_chunk_head(uint16_t num) {
    uint8_t size[6]; /* Chunk-size (four hex digits + CRLF). */
    uint8_t nibble; /* Nibble of @p num to convert. */
    int8_t i = 3; /* Index of @c size to write hex-digit. */

    size[5] = '\n';
    size[4] = '\r';

    while(i >= 0) {
        nibble = num & 0x0F;
        size[i] = nibble + (nibble < 10 ? '0' : 'A' - 10);
        num >>= 4;
        --i;
    }
}

return net_send(HTTP_SOCKET, size, 6, 0);
}

int16_t srvr_compile(uint8_t flush, ...) {
    int16_t outcome = 0; /* As returned from net_send(). */
    uint8_t buf[TXF_BUF_LEN]; /* Stores a fragment until it is sent. */
    uint8_t* str; /* Pointer to the string to actually send. */
    unsigned int txf_id; /* Value of any optional argument; txf index. */
    va_list ap; /* Optional argument reference. */
    uint8_t do_allcap = 0; /* Flag to make next fragment all upper-case. */
    uint8_t do_send; /* Flush flag of the current iteration. */

    va_start(ap, flush);
    txf_id = va_arg(ap, unsigned int);

    while(txf_id != SRVR_NOT_SET && outcome >= 0) {
        str = buf;
        do_send = 1;

```

```

/* Specify that all capitals is required and bring the next argument. */
if(txf_id == TXFx_TO_ALLCAP) {
    do_allcap = 1;
    txf_id = va_arg(ap, unsigned int);
}

/* If the fragment resides in program memory, fetch it from there. */
if(txf_id < TXF_MAX) {
    strcpy_P(buf, (PGM_P)pgm_read_word(&srvr_txf[txf_id]));
}

/* If the fragment resides in #server_consts, fetch it from there. */
} else if(txf_id == TXFx_FROMRAM) {
    txf_id = va_arg(ap, unsigned int);
    strcpy(buf, server_consts[txf_id]);

/* If a custom string is supplied, set @c str to simply point to it. */
} else if(txf_id == TXFx_FW_STRING) {
    str = va_arg(ap, uint8_t*);

/* If an integer is supplied, convert it into a string in @c buf, making
 * sure to set @c str to its first digit. */
} else if(txf_id == TXFx_FW_UINT) {
    uint8_t len; /* The number of bytes written. */
    len = uint_to_str(&buf[TXF_BUF_LEN - 1], va_arg(ap, uint16_t));
    str = &buf[TXF_BUF_LEN - 1 - len];

/* Ignore any invalid fragment IDs. */
} else {
    do_send = 0;
}

if(do_send) {

    if(do_allcap) {
        strupr(buf);
        do_allcap = 0;
    }

    outcome = net_send(HTTP_SOCKET, str, strlen(str), 0);
}

txf_id = (unsigned int)va_arg(ap, unsigned int);
}

/* Flush all buffered data, if so specified. This should be avoided in case
 * any of the mentioned text fragments could not be written, because, then,
 * the text would not be complete / correct. */

```

```
if(flush && outcome >= 0) outcome = net_send(0, buf, 0, 1);

va_end(ap);
return outcome;
}

void srvr_call() {
    uint8_t uri; /* ID or requested URI. */
    uint8_t methods; /* Available methods for requested URI. */
    HTTPRequest req; /* Request representation. */

    /* Always reset to s_next() as it may have been altered due to a different
     * transfer coding of the previous request. */
    stream_set_source(&s_next);
    json_set_source(&s_next);

    http_parse_request(&req);
    uri = req.uri;

    /* Initialise the response line with the HTTP version followed by a single
     * space. This should be followed by an appropriate status code, headers and
     * a message body as determined further below. */
    srvr_prep(TXF_HTTpv, TXF_SPACE);

    /* TODO: If multiple content types of incoming messages are to be supported,
     * the appropriate parser for each request should be set here, *before*
     * calling the resource handler, below. Currently, only JSON is supported and
     * its parser is set only once, during initialisation. See srvr_init() and
     * its line with rsrc_set_parser(). */

    /* If the URI is not available or if no handler is specified, return 404
     * (Not Found). */
    if(uri == SRVR_NOT_SET || !srvr.rsrc_handlers[uri].call) {
        srvr_send(TXF_STATUS_404, TXF_ln,
                  TXF_STANDARD_HEADERS_ln,
                  TXF_CONTENT_LENGTH_ZERO_ln, TXF_ln);
        return;
    }

    /* Method not recognised by the server or entity-body in a transfer-coding
     * it does not understand. Return 501 (Not Implemented). */
    if(req.method == SRVR_NOT_SET
    || req.transfer_encoding == TRANSFER_COD_OTHER) {
        srvr_send(TXF_STATUS_501, TXF_ln,
                  TXF_STANDARD_HEADERS_ln,
                  TXF_CONTENT_LENGTH_ZERO_ln, TXF_lnl);
        return;
    }
}
```

```

/* Call the handler, if the requested method has a bit-flag set. */
methods = srvr.rsrc_handlers[uri].methods;
if(TO_METHOD_FLAG(req.method) & methods) {

    /* Set-up reading a chunked message, if that was specified in the
     * header. */
    if(req.transfer_encoding == TRANSFER_COD_CHUNK) {
        stream_set_source(&c_next);
        json_set_source(&c_next);
    }

    /* Call the appropriate handler. */
    (*(srvr.rsrc_handlers[uri].call))(&req);

    /* Otherwise, return a 405 (Method Not Allowed), along with an 'Allow'
     * header. */
} else {
    uint8_t i;

    /* Send the initial headers. */
    srvr_prep(TXF_STATUS_405, TXF_ln,
              TXF_STANDARD_HEADERS_ln,
              TXF_CONTENT_LENGTH_ZERO_ln,
              TXF_ALLOW, TXF_HS);

    /* Loop and print all the available methods (in upper-case). */
    for(i = 0 ; i < METHOD_MAX ; ++i) {
        if(methods & 1) {
            srvr_prep(TXFx_TO_ALLCAP, TXFx_FROMRAM, METHOD_MIN + i);

            methods >>= 1;

            /* If there are more methods available, print the list-value
             * separator (comma) and proceed. */
            if(methods) srvr_prep(TXF_COMMA, TXF_SP);

        } else {
            methods >>= 1;
        }
    }
    srvr_send(TXF_lnl);

    return;
}
}

```

## Αναλυτής και γεννήτρια JSON

### json\_parser.h

```
/**  
 * @file  
 * @brief A rudimentary JSON parser.  
 * @addtogroup json_parser JSON Parser  
 * @ingroup http_server  
 * @  
 * @  
 * @brief A rudimentary JSON parser.  
 *  
 * @section sec_json_parser_reason The reason  
 * JSON parsing is required as this data format is used as a lightweight  
 * container to receive the parameters needed to perform operations on the  
 * exposed resources of the device through the use of HTTP. These parameters are  
 * to be supplied as member names of a serialized JSON object, the latter of  
 * which would constitute the entity-body of the request itself. The main  
 * requirements for such a parser (other than recognising the JSON @c object  
 * value), are to:  
 * — Parse input as it becomes available without the need to be provided a  
 * buffer upon which to operate.  
 * — Convert serialized data to appropriate data types for use by the  
 * application, specifying which were set and which were erroneous.  
 * — Produce as little overhead as possible (mainly as far as program space  
 * consumption is concerned).  
 *  
 * It was, thus, deemed necessary to design and implement a specific JSON parser  
 * for this application.  
 *  
 * @section sec_json_parser_current_status Current status  
 * Only limited support for serialized JSON formatted data is provided, loosely  
 * based on 'RFC 7159 — The JavaScript Object Notation \(JSON\) Data Interchange Format', as this implementation  
 * is not meant to be a full-fledged, general purpose parser; it is specifically  
 * designed to meet the current needs of the application with as little overhead  
 * as possible, even to the expense of conformance to that RFC.  
 *  
 * As a result, out of the seven defined JSON values (@c object, @c array, @c  
 * number, @c string, @c false, @c null and @c true), only objects, numbers and  
 * strings are recognised, liable to the following restrictions:  
 * — Objects are not nested (ie, no object is set as a member value within  
 * another object).  
 * — Object members contain a key of type @c string. All strings begin and end  
 * with quotation marks (RFC 7159 p.8).  
 * — A predefined set of possible member names (tokens) is provided along with  
 * their expected data type and storage memory (see param.h). Should the
```

```
* parser come across a member name, type or size other than those in the
* set, parsing stops.
* — Numbers are unsigned integers of specified resolution (8- or 16-bit).
* — Escaped Unicode characters in strings are not recognised. If present, they
* are preserved as a plain text.
*
* Other than those restrictions, it is irrelevant to the parser whether any of
* the specified members (tokens) were present, the order in which, or how many
* times they have occurred. The caller may identify whether a member was
* specified by checking its corresponding status bits (see ParamValue#status_len
* in param.h). If set multiple times, the value from the last member occurrence
* is preserved.
*
* According to the specification, any number of insignificant white-space may
* occur around a structural character, the latter being a sequence of a
* white-space followed by a reserved character and another white-space (*RFC
* 7159 p.5*), eg:@verbatim
begin-object = ws %x7B ws ; { left curly bracket
@endverbatim
* Any amount of leading or trailing white-space is ignored.
*
* This module is separated into more specific functions, each responsible for a
* different section of a serialised JSON object. All of them are strict in the
* sense they fail fast upon any unexpected occurrence. This lack of leniency
* provides greater minimalism and, thus, requires less program space. Successful
* completion is indicated by a return value of @c 0, whereas failure, by a
* negative value (typically, #OTHER or #EOF). These functions, of course, use
* one another. Currently, should any one of them fail will cause the others to
* fail, as well; #json_parse() -- the root function -- returns that value.
* In case of an error, the stream will not be advanced any further. The
* character that caused the failure, though, will not be reinstated back into
* the stream.
*
* @section sec_json_parser_use Use
* The predominant function of this module is json_parse() which will initiate
* parsing on the input stream. To use it, one must first set the input stream.
* This is done by calling json_set_source() at least once, supplying a function
* address. One should not attempt to call json_parse() without setting a valid
* function pointer. Failing to do so would compromise the entire application.
*
* @subsection sec_json_parser_dependencies Dependencies
* In order for this component to be operable, some additional components are
* required, such as the ability to compare an incoming string from the stream
* against the specified set of tokens (stream match), and a number of functions
* to convert serialized data to variable values (integer parser, and string
* copy). These requirements are satisfied by the following functions of
* stream_util.h:
* — stream_match(uint8_t** tokens, uint8_t max, uint8_t* c)
```

```
* — parse_uint8(uint8_t* value, uint8_t* c)
* — copy_until(uint8_t* buf, uint8_t delim, uint8_t max, uint8_t* c)
*
* Care must be taken than the input function of that module is set to the
* appropriate one *before* invoking json_parse().
*/

#ifndef JSON_PARSER_H_INCL
#define JSON_PARSER_H_INCL

#include "param.h"
#include "stream_util.h"

#include <inttypes.h>

/**
* @brief Represents the current stage during various levels of input parsing.
*/
enum JSONState {
    /** @brief An object opening section may have been found. */
    JSON_OBJECT_BEGIN,

    /** @brief An object closing section has been found. */
    JSON_OBJECT_END,

    /** @brief A member opening section may have been found. */
    JSON_MEMBER_BEGIN,

    /** @brief A member closing section has been found. */
    JSON_MEMBER_END,

    /** @brief A member key (token) start may have been found. */
    JSON_KEY_BEGIN,

    /** @brief A member key (token) end may have been found. */
    JSON_KEY_END,

    /** @brief A value opening section has been found. */
    JSON_VALUE_BEGIN,

    /** @brief A value has been parsed. */
    JSON_VALUE_END
};

/**
* @brief Check whether @p x is a JSON white-space character.
*
* @p x is compared against the four characters defined a white-space: space
```

```

* (0x20), horizontal tab (0x09), line feed (0x0A) and carriage return (0x0D).
* (*RFC 7159 p.5*)
*/
#define JSON_IS_WS(x) (x == ' ' || x == '\t' || x == '\n' || x == '\r')

/**
* @brief Sets the function that supplies this module with bytes from the input
* stream.
*
* Sets the value of #gnext.
* The provided function should accept a single byte address into which to store
* the next byte found on the stream. It should return @c 0 if the contents of
* that byte had been successfully updated or #EOF, if no more bytes are
* available on the stream for this particular processing cycle.
*
* It should give access to the serialized JSON object regardless of
* transformations that may have been applied to the actual input (eg,
* compression, chunked input, local buffering).
*
* Note that if any of the provided functions of this module are invoked without
* previously setting the function pointer will, in all probability, cause the
* application to fail.
*
* @param[in] input_source Pointer to input function.
*/
void json_set_source(int8_t (*input_source)(uint8_t*));

/**
* @brief Search the specified parameters on a JSON formatted input stream.
*
* For a list of limitations and features of the current implementation, refer to
* the details section.
*
* @param[in] tokens An array of parameter-tokens (strings) that are to be
* searched on the stream.
* @param[in,out] values An array of #ParamValue that describe the semantics
* around each string provided in @p tokens.
* @param[in] len The amount of elements in @p tokens and @c values (obviously,
* the same for both).
* @returns One of:
* — @c 0; if the stream was successfully parsed. The status bits of each
* #ParamValue in @p values, provide details for each parameter.
* — #OTHER; if parsing failed at any stage (eg, unexpected parameter-token).
* If an unacceptable value was provided for an acceptable token, its
* corresponding status bits are set (#PARAM_INVALID or #PARAM_TOO_LONG).
* — #EOF; if end-of-stream occurred prematurely while parsing the serialized
* input.
*/

```

```
int8_t json_parse(uint8_t** tokens, ParamValue* values, uint8_t len);

/**  
 * @brief Produce a serialised object of the provided parameters.  
 *  
 * @p tokens is an array of object keys that will be included in the serialised  
 * output. Each one will be enclosed in double quotes and followed by the value  
 * in the corresponding index of @p values. The conversion output depends on the  
 * specified #DataType. For #DTYPE_UINT, a fixed width sub-string of 5 characters  
 * is produced (which contains up to 5 digits and white-space leading padding).  
 * Currently, only 8-bit numbers are supported; in future versions, the size bits  
 * of @link ParamValue#status_len status_len@endlink would suffice to support  
 * greater resolutions.  
 * For #DTYPE_STRING, the contents of @link ParamValue#data_ptr data_ptr@endlink  
 * are copied (within double quotes) until the first occurrence of a null-byte.  
 *  
 * @an "atomic" is series of key-value pairs (which may be  
 * separated among different calls).  
 *  
 * Simple support is provided for serialising directives, as follows:  
 * - #SERIAL_FLUSH; flush data after serialising the supplied values.  
 * - #SERIAL_ATOMIC_S; the supplied values are the entry of an object and  
 * should be prefixed with a brace.  
 * - #SERIAL_ATOMIC_E; the supplied values are the tail of an object and should  
 * be followed by a brace.  
 * - #SERIAL_ENVELOPE_S; the first value in @p tokens is the key for an array.  
 * The corresponding index in @p values is ignored. This results in:  
 * @code  
 * "token": [  
 * - #SERIAL_ENVELOPE_E; terminate a previously initiated envelope (see  
 * previous bullet). The envelope may or may not have been initiated with  
 * this call. @p tokens and @p values is not accessed in this case.  
 * - #SERIAL_PRECEDED; a separator (comma) will be prefixed before serialising  
 * object key-pairs or an envelope start. It is ignored in case  
 * #SERIAL_ENVELOPE_E without #SERIAL_ENVELOPE_S has been specified.  
 *  
 * Examples:  
 * 1. Serialise an empty object, flushing after doing so: @verbatim  
 (NULL, NULL, 1, SERIAL_DEFAULT)  
 Produces:  
 {  
 }  
 @endverbatim  
 * 2. Serialise an object with two key-value pairs, flushing after doing so:  
 * @verbatim  
 (tokens, values, 2, SERIAL_ATOMIC_S | SERIAL_ATOMIC_E | SERIAL_FLUSH)  
 Produces:  
 {  
 "token1": 14,
```

```

    "token2": "string"
}
@endverbatim
* 3. Serialise the start of an object with one key–value pair. Do not send
* the result, yet: @verbatim
(tokens, values, 1, SERIAL_ATOMIC_S)
Produces:
{
    "token1": 14
}@endverbatim
* 4. Serialise the start of an array that contains one key–value pair.
* Designate that the array is preceded by another key–value pair. Do not
* send the result, yet: @verbatim
(tokens, values, 2, SERIAL_PRECEDED | SERIAL_ENVELOPE_S)
Produces:
,
"array": [
    "token3": "more strings"
}@endverbatim
* 5. Serialise the tail of an object that has an open envelope with a
* key–value pair. Send the result, afterwards: @verbatim
(tokens, values, 1, SERIAL_PRECEDED
    | SERIAL_ENVELOPE_E
    | SERIAL_ATOMIC_E
    | SERIAL_FLUSH)
Produces:
,
"token4": "and.. flush!"
]}
@endverbatim
*
* Note that examples 3 through 5 could be part of the same entity.
*
*
* @param[in] tokens An array of parameter–tokens (strings) that are to be
* used as object keys.
* @param[in] values An array of #ParamValue that describe the semantics around
* each string provided in @p tokens (#DataType and, perhaps, size).
* @param[in] len The amount of elements in @p tokens and @c values (obviously,
* the same for both).
* @param[in] ctr Serialising directives (see above).
*/void json_serialise(uint8_t** tokens,
                     ParamValue* values,
                     uint8_t len,
                     uint8_t ctr);

/**
* @brief Advance the stream till a non–white–space character.

```

```
*  
* In JSON, a white-space character is one of: space (0x20), horizontal tab  
* (0x09), line feed (0x0A) and carriage return (0x0D) (*RFC 7159 p.5*).  
*  
* @param[in,out] c The first character to start parsing from and the last one  
* read from the stream. When called, if it is not a white-space, it  
* immediately returns.  
* @returns One of:  
* - 0; if a non-white-space character was read from the stream.  
* - #EOF; if the end-of-stream has occurred.  
*/  
int8_t json_discard_WS(uint8_t* c);  
  
/**  
* @brief Parse the stream for a serialized JSON object.  
*  
* Nested objects are not supported at this time.  
*  
* When called, @p c should contain the left curly bracket '{' (0x7b) of the  
* object to parse ('begin-object'). Upon normal completion (return value @c 0),  
* the character pointed to by @c p is its corresponding right curly bracket '}'  
* (0x7d) ('end-object').  
*  
* This function uses #json_parse_member() which in turn uses  
* #json_parse_value(). As stated before, they are all strict against unexpected  
* occurrences. As far as this function is concerned, it means than all objects  
* must start and end with a pair of curly brackets, optionally containing any  
* number of members separated with *one* value-separator (*RFC 7159 p.6*):  
* @verbatim  
object = begin-object [ member *( value-separator member ) ]  
                  end-object  
  
member = string name-separator value@endverbatim  
*  
* What is more important, though, is that a @c member's key is a @c string, and  
* all strings start and end within quotation-marks (*RFC 7159 p.8*). As a  
* result, this function will fail (return value @c #OTHER) should a key be  
* specified without quotation-marks.  
*  
* Only keys specified in @p info may be accepted, the order of which is  
* irrelevant (see #json_parse_member()).  
*  
* @param[in,out] info Provides the acceptable keys and a additional information  
* (for an explanation, see #ParamInfo).  
* @param[in,out] c The first character to start parsing from and the last one  
* read from the stream. Upon invocation, it should point to '{' (0x7b).  
* @returns One of:  
* - @c 0; if the object was successfully parsed and no errors have occurred.
```

```

* — #OTHER; if an invalid character has occurred at any stage during
* processing.
* — #EOF; if end-of-stream was reached at any point.
*/
static int8_t json_parse_object(ParamInfo* info, uint8_t* c);

/**
* @brief Parses the stream for a member whose key is one of those found in @p
* info.
*
* When calling this, @p c should point to "" (double quote) to designate the
* start of a key (keys are defined as strings and are, thus, enclosed within
* quotation marks). The input stream is matched against the values included in
* @p info (member @link #ParamInfo::tokens tokens@endlink) using the externally
* provided
* stream_match()
* function in an attempt to locate one of those keys/tokens. If it succeeds, the
* return value (status) of this function depends on the outcome of parsing the
* identified key's value by #json_parse_value(). On the contrary, if it happens
* upon a key that is not included in @p info, the operation stops (fails) and
* #OTHER is returned.
*
* @param[in,out] info Provides the acceptable keys and a additional information
* (for an explanation, see #ParamInfo).
* @param[in,out] c The first character to start parsing from and the last one
* read from the stream. Upon invocation, it should point to "" (string).
* @returns One of:
* — @c 0; if a member (key/value pair) with a key defined in @p info was
* successfully parsed.
* — #OTHER; if an invalid character has occurred at any stage during
* processing.
* — #EOF; if end-of-stream was reached at any point.
*/
static int8_t json_parse_member(ParamInfo* info, uint8_t* c);

/**
* @brief Parse the stream for a value of type and size defined by @p pvalue.
*
* @c pvalue.@link ParamValue::type type@endlink is used to determine the
* appropriate value parser to call (currently, one of the externally supplied
* #parse_uint8() or #copy_until()). According to the return value of those
* functions, the two most-significant bits in @c
* pvalue.@link ParamValue::status_len status_len@endlink are set to one of
* #PARAM_INVALID, #PARAM_TOO_LONG or #PARAM_VALID. In case #PARAM_VALID is set,
* the actual value can be read from @c
* pvalue.@link ParamValue::data_ptr data_ptr@endlink. Do note that @c
* pvalue.@link ParamValue::data_ptr data_ptr@endlink is *not* allocated within
* this function but *should* have been set to point to a valid memory location

```

```
* *before* the execution of this function. For details, see #ParamValue.  
*  
* Also note that even though the status of @c  
* pvalue.@link ParamValue::status_len status_len@endlink may have been set to  
* #PARAM_INVALID or #PARAM_TOO_LONG, the data pointed to by @c  
* pvalue.@link ParamValue::data_ptr data_ptr@endlink  
* may have been altered during processing. In any case, they should still be  
* considered invalid.  
*  
* @param[in,out] pvalue Provides the type and size of the acceptable value as  
* well as access to its storage memory (for an explanation, see #ParamValue).  
* @param[in,out] c The first character to start parsing from and the last one  
* read from the stream.  
* @returns One of:  
* — @c 0; if a value in accordance to @p pvalue has been read from the stream  
* and stored into pvalue.@link ParamValue::data_ptr data_ptr@endlink.  
* — #OTHER; if an invalid character has occurred at any stage during  
* processing. This will also be reflected by #PARAM_INVALID being set into  
* pvalue.@link ParamValue::status_len status_len@endlink.  
* — #EOF; if end-of-stream was reached at any point.  
*/  
static int8_t json_parse_value(ParamValue* pvalue, uint8_t* c);  
  
#endif /* JSON_PARSER_H_INCL */  
/** @} */
```

### json\_parser.c

```
#include "json_parser.h"  
#include "w5100.h"  
#include "util.h"  
#include "defs.h"  
  
#include <stdio.h>  
#include <string.h>  
#include <inttypes.h>  
  
/**  
* @ingroup json_parser  
* @brief Function pointer to access the next character to parse.  
*  
* It provides this module's components access to the input stream. In a typical  
* application, this should be the same stream that supplies characters to  
* external helper functions (such as stream_match()). It is set using  
* json_set_source().  
*/  
static int8_t (*gnext)(uint8_t*);
```

```

void json_set_source(int8_t (*input_source)(uint8_t*)) {
    gnext = input_source;
}

int8_t json_parse(uint8_t** tokens, ParamValue* values, uint8_t len) {
    int8_t c_type; /* Operation status (return value). */
    uint8_t c; /* Passed from one stream parser function to another. */
    uint8_t i;

    /* A wrapper around tokens (strings) and ParamValues used by the underlying
     * API. */
    ParamInfo match = {.tokens = tokens, .values = values, .len = len};

    /* Reset the status bits of the params that are being searched for. */
    for(i = 0 ; i < len ; ++i) {
        values[i].status_len &= ~PARAM_STATUS_MASK;
    }

    /* Discard leading white-space from the stream. An initial white-space
     * character is faked by setting @c c to space. */
    c = ' ';
    c_type = json_discard_WS(&c);

    if(!c_type) {
        /* Only objects are supported at this time. If the content is not a
         * valid object, #OTHER will be returned. */
        c_type = json_parse_object(&match, &c);
    }

    return c_type;
}

void json_serialise(uint8_t** tokens,
                    ParamValue* values,
                    uint8_t len,
                    uint8_t ctr) {

    uint8_t buf[6]; /* A local buffer. */
    uint8_t* str = buf; /* String to send. */
    uint8_t k = 0; /* Number of bytes to send. */
    uint8_t i = 0; /* Iteration of @p tokens. */
    uint8_t j; /* Pad numbers with white-space. */
    uint8_t flush = 0; /* Flush data after sending them. */

    uint8_t state = JSON_OBJECT_BEGIN;

    while(len) {
        switch(state) {

```

```
case JSON_OBJECT_BEGIN:
    /* If there are previous values, prefix a comma. */
    if(SERIAL_PRECEDED & ctr) buf[k++] = ',';

    /* A brace signifies the start of an object. */
    if(SERIAL_ATOMIC_S & ctr) buf[k++] = 0x7b; /* '{' */

    buf[k++] = '\n';

    /* Initiate a key, if there are key tokens specified. */
    if(tokens != NULL) {
        buf[k++] = '\"'; /* Key-start. */
        state = JSON_KEY_BEGIN;
    } else {
        state = JSON_VALUE_END;
    }
break;

case JSON_KEY_BEGIN:
    /* Print token. A double quote has been previously sent. */
    str = tokens[i];
    k = strlen(str);
    state = JSON_KEY_END;
break;

case JSON_KEY_END:

    buf[k++] = '\"'; /* Key-end. */
    buf[k++] = ':';
    buf[k++] = '_';

    /* Print a bracket (start of array), if the key corresponds to
     * an envelope (array) and finish this iteration. */
    if(SERIAL_ENVELOPE_S & ctr) {
        buf[k++] = '[';
        state = JSON_VALUE_END;

    } else {
        /* Insert a double quote if a string is to be printed. */
        if(values[i].type == DTTYPE_STRING) {
            buf[k++] = '\"';
        }

        /* Next up, print string or number. */
        state = JSON_VALUE_BEGIN;
    }

    str = buf;
```

```

break;

case JSON_VALUE_BEGIN:
    switch(values[i].type) {
        case DTTYPE_UINT:
            /* Print a total of 5 digits (with padding to fill the
             * gaps, if needed). */
            k = 5;

            /* The size of padding. */
            j = 4 - uint_to_str(&buf[5],
                               *((uint8_t*)values[i].data_ptr));

            /* Pad with spaces to create a fixed-width number. */
            while(j) {
                buf[j] = ' ';
                --j;
            }
            buf[j] = ' ';

            str = buf;

            break;
        case DTTYPE_STRING:
            str = (uint8_t*)values[i].data_ptr;
            k = strlen(str);

            break;
    }
    state = JSON_VALUE_END;
break;

case JSON_VALUE_END:
    if(! (SERIAL_ENVELOPE_S & ctr)) {

        /* If a string was printed, append a closing quote. */
        if(values != NULL) {
            if(values[i].type == DTTYPE_STRING) buf[k++] = '"';
            }
        }

        ++i; /* Increase number of iteration. */
        --len; /* Decrease remainder of parameters. */

        /* Print the end of the envelope, if this was the last value. */
        if(SERIAL_ENVELOPE_E & ctr && len == 0) buf[k++] = ']';

        /* Append a comma, if more parameters follow unless this is the

```

```
* start of an envelope (eg, ''env'': [ ). Otherwise, append a
* brace at the end, if requested. */
if(len) {
    if(SERIAL_ENVELOPE_S & ctr) {
        /* Avoid printing an envelope start more than once. */
        ctr &= ~(SERIAL_ENVELOPE_S);
    } else {
        buf[k++] = ',';
    }

    buf[k++] = '\n';
    buf[k++] = '"'; /* Key-start. */
    state = JSON_KEY_BEGIN;

} else {
    if(SERIAL_ATOMIC_E & ctr) {
        buf[k++] = '\n';
        buf[k++] = 0x7d; /* } */
    }
    flush = SERIAL_FLUSH & ctr;
}

str = buf;
break;
}

net_send(HTTP_SOCKET, str, k, flush);

k = 0;
}
}

int8_t json_discard_WS(uint8_t* c) {
    int8_t c_type = 0;

    while(!c_type && JSON_IS_WS(*c)) {
        c_type = (*gnext)(c);
    }
    return c_type;
}

static int8_t json_parse_object(ParamInfo* info, uint8_t* c) {
    int8_t state = JSON_OBJECT_BEGIN;
    int8_t go_on = 1;
    int8_t c_type = 0;

    while(!c_type && go_on) {
        c_type = json_discard_WS(c);
```

```

    if(c_type == EOF) break;

    switch(state) {
        case JSON_OBJECT_BEGIN:
            if(*c == 0x7b) { /* { */
                c_type = (*gnext)(c);
                state = JSON_MEMBER_BEGIN;
            } else {
                c_type = OTHER;
            }
            break;
        case JSON_MEMBER_BEGIN:
            if(*c == '""') {
                c_type = json_parse_member(info, c);
                state = JSON_MEMBER_END;
            } else if(*c == 0x7d) { /* } */
                go_on = 0;
                /* state = JSON_OBJECT_END; */
            } else {
                c_type = OTHER;
            }
            break;
        case JSON_MEMBER_END:
            if(*c == ',') {
                c_type = (*gnext)(c);
                state = JSON_MEMBER_BEGIN;
            } else if(*c == 0x7d) { /* } */
                go_on = 0;
                c_type = 0;
                /* state = JSON_OBJECT_END; */
            } else {
                c_type = OTHER;
            }
            break;
        }
        return c_type;
    }

    static int8_t json_parse_member(ParamInfo* info, uint8_t* c) {
        int8_t c_type; /* Return value of various functions (incl this one). */
        int8_t state; /* The current state of processing. */
        int8_t match; /* Index of a matching token. */
        int8_t go_on = 1; /* @c 0 indicates member parsing has been completed. */

        if(*c == '""') {
            c_type = (*gnext)(c);
            state = JSON_KEY_BEGIN;
        }
    }
}

```

```
    } else {
        c_type = OTHER;
    }

    while(!c_type && go_on) {
        c_type = json_discard_WS(c);
        if(c_type == EOF) break;

        switch(state) {
            case JSON_KEY_BEGIN:
                c_type = stream_match(info->tokens, info->len, c);

                /* If #stream_match() found a candidate token (c_type > 0) with
                 * a proper terminator ('"', in this case), then an acceptable
                 * key has been detected. Discard white-space, colon and any
                 * further white-space before calling the value-parser. */
                if(*c == '"') {
                    match = c_type;
                    c_type = (*gnext)(c);
                    state = JSON_KEY_END;
                } else {
                    c_type = OTHER;
                }
                break;
            case JSON_KEY_END:
                if(*c == ':') {
                    c_type = (*gnext)(c);
                    state = JSON_VALUE_BEGIN;
                } else {
                    c_type = OTHER;
                }
                break;
            case JSON_VALUE_BEGIN:
                c_type = json_parse_value(&info->values[match], c);
                go_on = 0;
                /* state = JSON_VALUE_END; */
                break;
        }
    }

    return c_type;
}

static int8_t json_parse_value(ParamValue* pvalue, uint8_t* c) {
    int8_t c_type;

    /* Acceptable length (if string) or resolution (if uint) of value. */
    uint8_t size = pvalue->status_len & 0x3F;
```

```

/* Status to apply to this param; default is to assume it is valid. */
uint8_t status = PARAM_VALID;

switch(pvalue->type) {
    case DTTYPE_UINT:

        if(size == 8) {
            c_type = parse_uint8((uint8_t*)(pvalue->data_ptr), c);
        } else if(size == 16) {
/* c_type = parse_uint16((uint16_t*)(pvalue->data_ptr), c); */
        }

        /* 'parse_uint' returns #OTHER when there are more digits available
         * than the supported resolution. */
        if(c_type == OTHER) {
            status = PARAM_TOO_LONG;

            /* Check whether termination occurred due to invalid character. */
        } else if(!JSON_IS_WS(*c) && *c != ',' && *c != 0x7d /* } */ ) {
            status = PARAM_INVALID;
            c_type = OTHER;
        }
        break;

    case DTTYPE_STRING:
        /* In JSON, strings begin with a """. If there is not one, then it
         * is not a valid string. */
        if(*c != "") {
            status = PARAM_INVALID;
            c_type = OTHER;
            break;
        }

        /* Read the first character after """ and start copying into the
         * assigned buffer. */
        c_type = (*gnext)(c); if(c_type == EOF) break;
        c_type = copy_until((uint8_t*)(pvalue->data_ptr), "", size, c);

        /* 'copy_until' returns #OTHER upon failing to read the delimiter
         * before exceeding its allowed amount of characters. */
        if(c_type == OTHER) {
            status = PARAM_TOO_LONG;

            /* Read the character after the terminating """ to return it to the
             * callee. */
        } else {
            c_type = (*gnext)(c);
        }
}

```

```

        break;
}

/* Apply value conformance status to it. Note that the two MSB are used to
 * that end. */
pvalue->status_len |= status;
return c_type;
}

```

## Ημερολόγιο μετρήσεων

### log.h

```

/**
 * @file
 * @addtogroup log Measurement Log
 * @brief Manage measurement logging.
 *
 * The records are stored within EEPROM in a circular structure that,
 * essentially, is a simplified circular buffer. Its contents span from address
 * #LOG_BASE_ADDR for a total of #LOG_LEN records. The size and contents of each
 * record is determined by #LogRecord.
 *
 * The chosen structure permits adding new records that, once the available space
 * has been depleted, replace the oldest ones. To achieve this, the offset of the
 * oldest record and the current amount of records are maintained (see #log).
 * Obviously, this offset is incremented as older records are being replaced.
 * Offsets that are calculated based on this start offset are referred to as
 * 'physical offsets' because they may be used to obtain the physical address of
 * a particular record.
 *
 * To avoid the implementation specifics and the manipulation of a circular
 * structure in higher abstraction functions, such as log_get_set() and
 * log_get_next(), they are designed to treat the Log as a linear structure where
 * the record at offset @c 0 is always the oldest one and the one at
 * (@link #log log.count@endlink - 1), the newest. These offsets are referred to
 * as 'logical offsets' and must first be mapped to a physical one before
 * accessing the corresponding record. This mapping is achieved by
 * log_get_offset().
 *
 * @{
 */
#ifndef LOG_H_INCL
#define LOG_H_INCL

#include "defs.h"

#include <inttypes.h>

```

```

/**
 * @brief Shorthand to calculate the address of a physical offset.
 */
#define LOG_ADDR(offset) (LOG_BASE_ADDR + offset*sizeof(LogRecord))

/**
 * @brief Index of a single record and the total amount of records.
 *
 * This is returned by log_get_set() and may be used with log_get_next() to
 * access the available Log records or a subset of them.
 *
 * It is also used as a reference point for the state of the circular Log
 * structure. See #log.
 */
typedef struct {
    /** @brief Index of some record.
     *
     * It spans from @c 0 up to #LOG_LEN - 1.
     */
    uint8_t index;

    /** @brief Amount of records. */
    uint8_t count;
} LogRecordSet;

/**
 * @brief Record structure.
 *
 * Note that the first member must be #BCDDate. This helps to avoid loading a
 * record in its entirety when searching for a particular date.
 */
typedef struct {
    /** @brief Date of record. Must be unique among all records. */
    BCDDate date;

    /** @brief Abscissa of sample coordinates. */
    uint8_t x;

    /** @brief Ordinate of sample coordinates. */
    uint8_t y;

    /** @brief Temperature of sample. */
    uint8_t t;

    /** @brief Relative humidity of sample. */
    uint8_t rh;
}

```

```
/** @brief pH of sample. */
uint8_t ph;
} LogRecord;

/** 
* @brief Initialise Log dependencies.
*
* It loads @c index and @c count from EEPROM into #log.
*/
void log_init();

/** 
* @brief Remove records newer than @p dt.
*
* This simply updates (decreases) an internal counter; the actual records are
* not erased from the EEPROM but are ignored and successively overwritten with
* each call to log_append().
*
* @param[in] dt The starting date. Records with a date equal or greater than
* this value, will be purged.
* @returns The number of records deleted.
*/
uint8_t log_purge(BCDDate* dt);

/** 
* @brief Add a new log record.
*
* Apart from writing the record, it also updates @c index, @c count (in EEPROM)
* and #log, as needed.
*
* @param[in] rec The record to append to the log.
*/
void log_append(LogRecord* rec);

/** 
* @brief Advance @p set to skip an @p amount of records.
*
* @param[in,out] set #LogRecordSet to update.
* @param[in] amount Amount of records to skip.
* @returns The amount of available records after skipping.
*/
uint8_t log_skip(LogRecordSet* set, uint8_t amount);

/** 
* @brief Read the next record found in the record @p set.
*
* Each successive call to this function reads one more record into @p rec. @c -1
* is returned when there are no more records available.
*/
```

```

*
* A valid #LogRecordSet is obtained via log_get_set(). As @p set is modified
* internally, any external modification could cause unexpected behaviour and
* should be avoided.
*
* @param[in] rec Contents of the next record.
* @param[in,out] set Set of records to return.
* @returns @c 0, if a record has been loaded into @p rec; @c -1, if there are no
* more records available (in which case, the contents of @p rec are *not*
* updated).
*/
uint8_t log_get_next(LogRecord* rec, LogRecordSet* set);

/**
* @brief Give a description of records that span between two dates.
*
* It initialises @p set so it may later be used to extract the desired amount of
* records that lay within the two specified dates. @p set should be a valid
* variable address and not @c NULL. Once initialised, @p set may be used with
* log_get_next() to retrieve each record. The contents of @p set are meaningful
* to and altered by the underlying API and should not be tampered with.
*
* @param[out] set A valid #LogRecordSet variable address to initialise.
* @param[in] since The starting date of the returned records (inclusive).
* @param[in] until The ending date of the returned records (inclusive).
* @returns Amount of records to be returned with this set.
*/
uint8_t log_get_set(LogRecordSet* set, BCDDate* since, BCDDate* until);

/**
* @brief Locate the closest record index to the supplied date.
*
* It implements a simple binary search algorithm to avoid unnecessary EEPROM
* reads. It uses <string.h>memcmp() for the date comparisons. If a record with
* the specified date is not found, the closest logical offset is returned,
* instead.
*
* This function makes the assumption that each record begins with a #BCDDate (or
* equivalent) data structure and only reads that many bytes.
*
* @param[out] index The logical offset of the closest matching record date to
* @p q.
* @param[in] q The date of the record in question.
* @returns The output of memcmp() of the last comparison.
*/
static int16_t log_find(uint8_t* index, BCDDate* q);

/**

```

```
* @brief Translate a logical to a physical offset.  
*  
* Physical offsets are used to access a record within the storage structure.  
* Logical offsets refer to a conceptual array where the oldest record is always  
* found at offset @c 0. This function translates a logical to a physical offset.  
*  
* @param[in] index A value between @c 0 and #LOG_LEN - 1. Otherwise, an offset  
* that lies outside the storage may be returned.  
* @returns The physical offset that may be used to access a record.  
*/  
static uint8_t log_get_offset(uint8_t index);  
  
#endif /* LOG_H_INCL */  
/** @} */
```

**log.c**

```
#include "log.h"  
#include "defs.h"  
#include <avr/eeprom.h>  
  
#include "string.h"  
  
/**  
* @brief Avoid first byte.  
*  
* This is to ensure that no sensitive data will be stored on the first EEPROM  
* byte.  
*/  
uint8_t eeprom_dummy EEMEM;  
  
/**  
* @brief Offset of the first stored record.  
*  
* It spans from @c 0 up to #LOG_LEN - 1.  
*/  
static uint8_t log_index EEMEM = 0;  
  
/**  
* @brief The amount of stored records.  
*/  
static uint8_t log_count EEMEM = 0;  
  
/**  
* @ingroup log  
* @brief Internal Log state.  
*  
* @link LogRecordSet#index .index@endlink is the offset of the oldest record  
* within the circular structure. Due to the nature of the storage structure, the
```

```

* oldest record may reside anywhere between @c 0 and #LOG_LEN - 1.
* log_get_offset() uses this member to map a logical offset to a physical one.
*
* @link LogRecordSet#count .count@endlink is the amount of valid Log records.
*/
static LogRecordSet log;

void log_init() {
    log.index = eeprom_read_byte(&log_index);
    log.count = eeprom_read_byte(&log_count);
}

uint8_t log_purge(BCDDate* dt) {
    uint8_t count;
    LogRecordSet set;
    BCDDate until = {.year = 0x99, .mon = 0x12, .date = 0x31,
                      .hour = 0x23, .min = 0x59, .sec = 0x59};

    /* Find records between @p dt and end-of-time. */
    count = log_get_set(&set, dt, &until);
    if(count) {
        log.count -= count;
        eeprom_write_byte(&log_count, log.count);
    }

    DBG.printf("Purged records: %d\n", count);
    return count;
}

void log_append(LogRecord* rec) {
    uint8_t write_offset; /* Offset from #LOG_BASE_ADDR to write to. */

    /* Remove any records with a newer date than the one in @p rec. */
    /*log_purge(&rec->date);*/

    /* If the storage is full, replace the oldest record with this one. */
    if(log.count == LOG_LEN) {
        write_offset = log.index;

        /* Update the physical offset of the oldest record. */
        log.index = log.index == LOG_LEN - 1 ? 0 : log.index + 1;
        eeprom_write_byte(&log_index, log.index);

    } else {
        write_offset = log_get_offset(log.index + log.count);

        /* Update the count of available records. */
        ++log.count;
    }
}

```

```
    eeprom_write_byte(&log_count, log.count);
}

/* Calculate the physical address that corresponds to @c write_offset and
 * write to it. */
eeprom_update_block(rec, (void*)LOG_ADDR(write_offset), sizeof(LogRecord));
}

uint8_t log_skip(LogRecordSet* set, uint8_t amount) {
    /* Ensure there are enough records to skip. */
    if(set->count > amount) {
        set->index -= amount;
        set->count -= amount;

        /* Otherwise, specify the set is empty. */
    } else {
        set->count = 0;
    }
    return set->count;
}

uint8_t log_get_next(LogRecord* rec, LogRecordSet* set) {
    uint8_t read_offset;

    /* Read the next record provided there is one. */
    if(set->count && set->index < log.count) {
        read_offset = log_get_offset(set->index);

        eeprom_read_block(rec, (void*)LOG_ADDR(read_offset), sizeof(LogRecord));

        --(set->index);
        --(set->count);

        return 0;
    }
    return -1;
}

uint8_t log_get_set(LogRecordSet* set, BCDDate* since, BCDDate* until) {
    uint8_t i_since; /* Index of date @p since. */
    uint8_t i_until; /* Index of date @p until. */
    int16_t c_since; /* Comparison result of date @p since. */
    int16_t c_until; /* Comparison result of date @p until. */

    set->count = 0;

    if(log.count == 0) return 0;
```

```

/* Return the empty set, for improper date range. */
if(memcmp(since, until, sizeof(BCDDate)) > 0) {
    return 0;
}

/* Find the closest matching index for each date. */
c_since = log_find(&i_since, since);
c_until = log_find(&i_until, until);

/* Avoid counting records for the empty set. An empty set occurs when the
 * upper limit is lower than the lower limit or when both
 * upper and lower limits point at the same index and their respective
 * dates are both either greater or less than the date at that index. */
if(i_since == i_until &&
(c_since & 0x80) == (c_until & 0x80) && c_since != 0 && c_until != 0) {

/* Determine whether the limits need to be adjusted. */
} else {

/* If date @p since is greater than the date at the returned index, that
 * date must not be included in the set (increase lower limit). */
if(c_since > 0 && i_since < log.count - 1) ++i_since;

/* Likewise, for date @p until (decrease upper limit). */
if(c_until < 0 && i_until > 0) --i_until;

set->index = i_until;
set->count = i_until - i_since + 1;
}

return set->count;
}

static int16_t log_find(uint8_t* index, BCDDate* q) {
    int16_t start = 0; /* Sub-array lower search limit. */
    int16_t end = log.count - 1; /* Sub-array upper search limit. */

    BCDDate dt; /* Loaded record date. */
    int16_t cmp; /* Comparison result. */

    while(end >= start) {
        *index = start + (end - start)/2;

        /* Load date for the physical offset that corresponds to @c i. */
        eeprom_read_block(&dt,
                         (void*)LOG_ADDR(log_get_offset(*index)),
                         sizeof(BCDDate));
    }
}

```

```

    cmp = memcmp(q, &dt, sizeof(BCDDate));

    if(cmp < 0) {
        end = *index - 1;

    } else if(cmp > 0) {
        start = *index + 1;

    } else {
        break;
    }
}

return cmp;
}

static uint8_t log_get_offset(uint8_t index) {
    uint8_t offset;

    /* The requested item lies within #log.index and LOG_LEN - 1. */
    if(LOG_LEN - log.index > index) {
        offset = log.index + index;

        /* The requested item lies within @c 0 and log.index. */
    } else {
        offset = index - (LOG_LEN - log.index);
    }

    return offset;
}

```

## Αρχείο εκκίνησης

### mcu.h

```

/**
@file
*/
#include <stdio.h> /* e.g. FILE, stdin, fdev_setup_stream() */

/** 
* @brief Handler for sending a single @c char over the USART.
*
* @param c The character to send
* @param stream Pointer to the output stream, as required by the underlying API.
* @returns Always returns 0 (success).
*/
int usart_putchar(char c, FILE* stream);

```

```

/**
 * @brief Handler for receiving a single @c char over the USART.
 *
 * @param stream Pointer to the input stream, as required by the underlying API.
 * @returns The character read.
 */
int usart_getchar(FILE* stream);

/**
 * @brief Initialise the various modules.
 *
 * It should be noted that this function sets the motor operating range
 * (motor_set_max()). As a result, upon completion, the motors will be resetting.
 */
static void init();

/**
 * @brief Modifies the system clock to the desired frequency.
 */
void init_clock();

/**
 * @brief Sets up the USART registers.
 */
void init_usart();

```

**mcu.c**

```

/**
@file
 */

#include "mcu.h"
#include "defs.h"

#include "net.h"
#include "http_server.h"
#include "resource.h"
#include "w5100.h"

#if defined (ENABLE_SERIAL_IO) && !defined (ENABLE_DEBUG)
#include "flash.h"
#endif

#include "task.h"
#include "motor.h"
#include "twi.h"
#include "rtc.h"

```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#include <util/delay.h>
#include <inttypes.h>

#ifndef ENABLE_SERIAL_IO
/** 
 * @brief Output stream over the USART.
 *
 * This stream is configured as the default output and error streams (stdout,
 * stderr).
 */
FILE usart_output = FDEV_SETUP_STREAM(usart_putchar, NULL, _FDEV_SETUP_WRITE);

/** 
 * @brief Input stream over the USART.
 *
 * This stream is configured as the default input stream (stdin).
 */
FILE usart_input = FDEV_SETUP_STREAM(NULL, usart_getchar, _FDEV_SETUP_READ);
#endif /* defined (ENABLE_SERIAL_IO) */

/** 
 * @brief Initialise MCU and resets all hardware.
 */
int main() {

    /** The tasks are outlined below: */

    /* Disable all interrupts during this procedure. */
    cli();

    /* Always disable the Watchdog Timer, even if it is not used (*Atmel p.52*).
     * If set, bit @c WRDF of @c MCUSR overrides bit @c WDE of @c WDTCR, so it
     * needs to be cleared first (*Atmel p.55*). */
    MCUSR &= ~_BV(WDRF);

    /* To change @c WDE (and/or the prescaler bits), bit @c WDCE of WDTCR must
     * be set first and the change must occur within four cycles, after which,
     * @c WDCE resets (*Atmel p.55*). What is not mentioned, though, is that bit
     * @c WDE must also be set. */
    WDTCR = _BV(WDCE) | _BV(WDE);
    WDTCR = _BV(WDCE);

    /** — Setup CPU clock. */
    init_clock();
}
```

```

#define ENABLE_SERIAL_IO

/** — Setup USART prescaler and enable receiver and transmitter. */
init_usart();

/** — Setup I/O streams. */
stdout = &usart_output;
stdin = &usart_input;
#endif

_delay_ms(2000);

/* When in Master SPI mode, if SS is input low, MSTR bit will be cleared. */
DDRB |= _BV(DDB2);

/** — Enable external interrupts on INT1 (on low level, by default). *Atmel
* p.72* */
EIMSK |= _BV(INT1);

/* Pins connected to @c nCS, @c S0 and @c S1 of @c MUX are used as output,
* while the pin connected to @c 2Z is used as input. Pins for the optical
* encoder enable are also set. XZ—motor enable pins need not be set as
* output as well because they are physically connected to @c S0 and @c S1 of
* @c MUX. */
MUX_nCS_DDR |= _BV(MUX_nCS);
MUX_S0_DDR |= _BV(MUX_S0);
MUX_S1_DDR |= _BV(MUX_S1);
MUX_2Z_DDR &= ~_BV(MUX_2Z);

/* Have MUX disabled, by default. */
MUX_DISABLE();

/* SCLK, MOSI */
DDRB |= _BV(DDB5) | _BV(DDB3);
DDRD |= _BV(DDD7);

/* Output to control nCS of Flash. */
DDRD |= _BV(DDD1);
PORTD |= _BV(PORTD1);

/* Initialise the remaining modules. */
init();

set_sleep_mode(_BV(SM1));

/* Set both Change Enable *and* System Reset Mode bits to enable setting the
* timeout. Then, the WDT is set to Interrupt (only) mode. This way, at
* #WDT_TIMEOUT intervals, the CPU will be woken from power-down mode. The

```

```
* WDT ISR checks whether sampling should be initiated. Once execution
* returns to the main loop, the CPU goes to power-down, again. The CPU maybe
* woken at any time by other sources, as well, such as a limit switch and/or
* an incoming HTTP request. Those requests could delay the CPU for as long
* they require with no fear of the WDT timeout; since the System Reset Mode
* is not activated, the WDT Interrupt will simply be queued, if it occurs
* before any previously activated ISR returns. */
WDTCSR = _BV(WDCE) | _BV(WDE);
WDTCSR = _BV(WDCE) | _BV(WDIE) | (WDT_TIMEOUT & (_BV(WDP3)
                                         | _BV(WDP2)
                                         | _BV(WDP1)
                                         | _BV(WDPO)));
sei();
while(1) {
    if(!task_pending()) {
        sleep_enable();
        sleep_cpu();
        sleep_disable();
    }
}

return 0;
}

void sys_get(uint8_t setting, void* value) {
    switch(setting) {
        case SYS_IADDR:
            net_read(NET_SIPR, value, 4);
            break;
        case SYS_GATEWAY:
            net_read(NET_GAR, value, 4);
            break;
        case SYS_SUBNET:
            net_read(NET_SUBR, value, 4);
            break;
        case SYS_HADDR:
            net_read(NET_SHAR, value, 6);
            break;
        case SYS_MTR_MAX:
            motor_get_max(value);
            break;
    }
}
```

```

    case SYS_TASK:
        task_get(value);
        break;
    }
}

int8_t sys_set(uint8_t setting, void* value) {
    int8_t ret = -1;

    switch(setting) {
        case SYS_IADDR:
            net_write(NET_SIPR, value, 4);
            ret = rtc_write(SYS_IADDR, value, 4);
            srvr_set_host_name_ip(value);

            break;
        case SYS_GATEWAY:
            net_write(NET_GAR, value, 4);
            ret = rtc_write(SYS_GATEWAY, value, 4);

            break;
        case SYS_SUBNET:
            net_write(NET_SUBR, value, 4);
            ret = rtc_write(SYS_SUBNET, value, 4);

            break;
        case SYS_HADDR:
            net_write(NET_SHAR, value, 6);
            ret = rtc_write(SYS_HADDR, value, 6);

            break;
        case SYS_MTR_MAX:
            motor_set_max(value);
            ret = rtc_write(SYS_MTR_MAX, value, 3);
            break;
        case SYS_TASK:
            task_set(value);
            ret = rtc_write(SYS_TASK, value, 2);
            break;
    }

    return ret;
}

static void init() {
    uint8_t settings[] = {FACTORY_IADDR,
                         FACTORY_GATEWAY,

```

```
FACTORY_SUBNET,  
FACTORY_HADDR,  
GRID_X_LEN, GRID_Y_LEN, GRID_Z_LEN,  
0, 0}; /* Task defaults are to disable it. */  
  
Position max;  
Task task;  
  
uint8_t rtc_sec;  
rtc_read(0, &rtc_sec, 1);  
  
/* If bit #RTC_CH is set, then the clock is not running. The implementation  
* never stops the RTC from running, so it may just be the battery supply was  
* removed to reset to factory (default) settings. */  
if(bit_is_set(rtc_sec, RTC_CH)) {  
  
    /* Write factory settings into the RTC memory. */  
    rtc_write(RTC_BASE, settings, SYS_SIZE);  
  
    /* Load battery-backed (RTC) settings. */  
} else {  
  
    rtc_read(RTC_BASE, settings, SYS_SIZE);  
}  
  
max.x = settings[SYS_MTR_MAX_X - RTC_BASE];  
max.y = settings[SYS_MTR_MAX_Y - RTC_BASE];  
max.z = settings[SYS_MTR_MAX_Z - RTC_BASE];  
  
task.interval = settings[SYS_TASK_INT - RTC_BASE];  
task.samples = settings[SYS_TASK_SAMPL - RTC_BASE];  
  
/* Network module */  
/* Setup buffer size. #HTTP_SOCKET is configured to 8KB on Tx and Rx). */  
net_socket_init(NET_SIZEn(HTTP_SOCKET, NET_SIZE_8),  
                NET_SIZEn(HTTP_SOCKET, NET_SIZE_8));  
  
/* Pass server settings to the W5100 and the HTTP server module. */  
srvr_set_host_name_ip(&settings[SYS_IADDR - RTC_BASE]);  
net_write(NET_SIPR, &settings[SYS_IADDR - RTC_BASE], 4);  
net_write(NET_GAR, &settings[SYS_GATEWAY - RTC_BASE], 4);  
net_write(NET_SUBR, &settings[SYS_SUBNET - RTC_BASE], 4);  
net_write(NET_SHAR, &settings[SYS_HADDR - RTC_BASE], 6);  
  
/* Mode register (MR) defaults look OK. The same applies for RTR (200ms  
* intervals) and RCR (8 retries). */  
  
/* Enable interrupts on HTTP_SOCKET. */  
net_write8(NET_IMR, NET_IR_Sn(HTTP_SOCKET));
```

```

/* Setup #HTTP_SOCKET for HTTP (TCP on port #HTTP_PORT). */
net_socket_open(HTTP_SOCKET, NET_Sn_MR_TCP, HTTP_PORT);
net_write8(NET_Sn_CR(HTTP_SOCKET), NET_Sn_CR_LISTEN);

/* Other modules; complementary ones, first. */
rsrc_init();
srvr_init();
log_init();
task_init();
task_set(&task);

motor_init();

/* Set operating range *after* the motors have been initialised. */
motor_set_max(&max);
}

void init_clock() {
    /* Clock Prescaler Change Enable bit (CLKPCE) of CLKPR must first be set
     while all other bits are cleared. */
    CLKPR = _BV(CLKPCE);

    /* Then, within four clock cycles, the appropriate prescaler bits of the
     same register are set while the CLKPCE bit is cleared. For a clock
     frequency of 4MHz, only CLKPS1 needs to be set. Atmel pp.34—37. */
    CLKPR = _BV(CLKPS1);
}

#ifndef ENABLE_SERIAL_IO

ISR(USART_RX_vect) {
    #ifndef ENABLE_DEBUG
    uint8_t buf[256];
    uint16_t page;
    uint16_t len;
    uint16_t i;

    page = usart_getchar(stdin);
    page = ((uint16_t)usart_getchar(stdin) << 8) | page;

    len = usart_getchar(stdin);
    len = ((uint16_t)usart_getchar(stdin) << 8) | len;

    /* Load data into a local buffer. */
    for(i = 0 ; i < len ; ++i)
        buf[i] = usart_getchar(stdin);
}

```

```

/* Wait until the Flash has completed all previous write operations. */
fls_wait_WIP();

/* Enable Latch and send data to the Flash. */
fls_command(FLS_WREN, NULL);
fls_exchange(FLS_WRITE, page, buf, len);

fls_wait_WIP();
#endif /* ! defined (ENABLE_DEBUG) */
}

int usart_putchar(char c, FILE* stream) {
    loop_until_bit_is_set(UCSROA, UDRE0);
    UDR0 = c;
    loop_until_bit_is_set(UCSROA, UDRE0);
    return 0;
}

int usart_getchar(FILE* stream) {
    loop_until_bit_is_set(UCSROA, RXC0);
    return UDR0;
}

void init_usart() {
    /* Set UBRn value. */
    UBRROH = (unsigned char)(UBRR_VALUE>>8);
    UBRROL = (unsigned char)(UBRR_VALUE);

    /* Set character size to 8 bits. */
    UCSROC = _BV(UCSZ01) | _BV(UCSZ00);

    /* Enable Rx-complete interrupts, Receiver and Transmitter. */
    UCSROB = _BV(RXCIE0) | _BV(RXENO);

#ifndef ENABLE_DEBUG
    UCSROB |= _BV(TXENO);
#endif
}
#endif /* defined (ENABLE_SERIAL_IO) */

```

### Τποσύστημα κίνησης

#### motor.h

```

/**
 * @file
 * @addtogroup motor
 * @{

```

```
*/  
  
#ifndef MOTOR_H_INCL  
#define MOTOR_H_INCL  
  
#include "defs.h"  
#include <inttypes.h>  
  
/**  
 * @brief Denotes the direction of motion.  
 *  
 * Also, see #MTR_X_INC.  
 */  
typedef enum {  
    MTR_INC = 0,  
    MTR_DEC = -1  
} MotorDir;  
  
/**  
 * @brief The available device space over axis X.  
 */  
#define GRID_X_LEN (10)  
  
/**  
 * @brief The available device space over axis Y.  
 */  
#define GRID_Y_LEN (11)  
  
/**  
 * @brief The available device space over axis Z.  
 */  
#define GRID_Z_LEN (4)  
  
/**  
 * @brief The time it takes to move one unit on any axis.  
 *  
 * In seconds.  
 */  
#define MTR_UNIT_TIME 1  
  
/**  
 * @brief Activates the PWM lock, disabling signal propagation when @c OCOA is  
 * disconnected from pin #MTR_nLOCK.  
 *  
 */  
#define LOCK_ENABLE() MTR_nLOCK_PORT &= ~_BV(MTR_nLOCK)  
  
/**
```

```
* @brief Disables the PWM lock, allowing its propagation to the motors when
* @c OC0A is disconnected from pin #MTR_nLOCK.
*/
#define LOCK_DISABLE() MTR_nLOCK_PORT |= _BV(MTR_nLOCK)

/**
* @brief Enables generation of PWM signal on pin #MTR_Y by connecting @c OC1A to
* the port.
*
* Mode of operation is Set/Clear on compare-match. *Atmel p.132.*
*/
#define PWM_Y_ENABLE() TCCR1A |= _BV(COM1A1)

/**
* @brief Disables generation of PWM signal on pin #MTR_Y by disconnecting @c
* OC1A from the port.
*/
#define PWM_Y_DISABLE() TCCR1A &= ~_BV(COM1A1)

/**
* @brief Enables generation of PWM signal on pin #MTR_XZ by connecting @c OC1B
* to the port.
*
* Mode of operation is Set/Clear on compare-match. *Atmel p.132.*
*/
#define PWM_XZ_ENABLE() TCCR1A |= _BV(COM1B1)

/**
* @brief Disables generation of PWM signal on pin #MTR_Y by disconnecting @c
* OC1A from the port.
*/
#define PWM_XZ_DISABLE() TCCR1A &= ~_BV(COM1B1)

/**
* @brief Condition to determine whether any Timer/Counter1 prescaler bits are
* set.
*/
#define PWM_IS_ON() (TCCR1B & (_BV(CS12) | _BV(CS11) | _BV(CS10)))

/**
* @brief Activates PWM generation.
*
* This macro should be called once:
* — The desired duty cycle has been set in @c OCR1A and/or @c OCR1B (for motor Y
* and XZ, respectively).
* — @c OC1A and/or @c OC1B has been connected to the port (#PWM_XZ_ENABLE() and
* #PWM_Y_ENABLE()).
* — An optical encoder has been selected (#MTR_ROUTE_X, #MTR_ROUTE_Z() or
```

```

* #MTR_ROUTE_Y()).
* — The multiplexer has been chip-selected (#MUX_ENABLE()).
*/
#define MTR_PWM_START() TCCR1B |= MTR_PRESCALER

/***
* @brief Stop motor PWM generation.
*
* Bits @c CS12, @c CS11 and @c CS10 of TCCR1B are zeroed-out.
*/
#define MTR_PWM_STOP() \
TCCR1B &= ~(_BV(CS12) \
            | _BV(CS11) \
            | _BV(CS10))

/***
* @brief Route signals from and to motor X.
*
* Sets the appropriate signals so that the PWM signal on #MTR_XZ pin is
* propagated to motor X instead of motor Z (the two being multiplexed on the
* same pin). This automatically enables the corresponding rotary encoder, the
* steps of which can be seen on #MUX_2Z pin.
*/
#define MTR_ROUTE_X() \
MUX_SO_PORT &= ~_BV(MUX_SO); \
MUX_S1_PORT |= _BV(MUX_S1)

/***
* @brief Route signals from and to motor Y.
*
* Enables the rotary encoder of motor Y, the steps of which can be seen on
* #MUX_2Z pin. Unlike motors X and Z, simply disabling the AutoLock while
* starting PWM generation is sufficient to set this motor in motion.
* Consequently, if running this motor along with another (and in particular
* motor X), calling this macro could/should be omitted.
*/
#define MTR_ROUTE_Y() \
MUX_SO_PORT &= ~_BV(MUX_SO); \
MUX_S1_PORT &= ~_BV(MUX_S1)

/***
* @brief Route signals from and to motor Z.
*
* Sets the appropriate signals so that the PWM signal on #MTR_XZ ping is
* propagated to motor Z instead of motor X (the two being multiplexed on the
* same pin). This automatically enables the corresponding rotary encoder, the
* steps of which can be seen on #MUX_2Z pin.
*/

```

```
#define MTR_ROUTE_Z() \
MUX_SO_PORT |= _BV(MUX_SO);\
MUX_S1_PORT &= ~_BV(MUX_S1)

/** \
* @brief Condition to determine whether a Y-axis limit switch has been engaged.
*/
#define IS_LMT_nY() ((LMT_nY_PIN & _BV(LMT_nY)) == 0)

/** \
* @brief Condition to determine whether an X- or Z-axis limit switch has been
* engaged.
*/
#define IS_LMT_nXZ() ((LMT_nXZ_PIN & _BV(LMT_nXZ)) == 0)

/** \
* @brief Velocity setting for OCR1B to move along the positive direction on axis
* X.
*
* With a total of 5000 increments in Timer/Counter1, a PWM signal with,
* approximately, 5 to 10% duty cycle is produced with pulses that remain high
* for 250 to 500 increments, respectively. Pulses below 7.5% (ie, from 250 up to
* approx. 375) result in a Counter-Clockwise (CCW) rotation; the lower the
* value, the greater the angular velocity. Similarly, increments in the range
* 375 up to 500 produce Clockwise (CW) rotation. In this case, the greater the
* value, the greater the angular velocity, as well. Increments around 375 force
* the motor to retain its position (brakes).
*
* In practice, the values around the middle point (7.5% duty cycle) are *not*
* symmetric, in the sense that, for example, 350 and 400 do *not* necessarily
* produce angular velocities of the same magnitude. The exact behaviour is
* strictly dependent on each motor and could differ even among motors of the
* same model. In the current configuration, calibration tests have been
* performed to determine duty cycles that result in similar velocities for axes
* X and Y. This was deemed necessary to ensure smooth operation when motion in
* both these axes is requested (see #setup_axis() and #setup_lock()).
*
* While the motors are rotating, the belt and pinion linear actuator converts
* rotational to linear motion in an either increasing or decreasing direction;
* it is the positioning of the motors that determines what CW and CCW rotation
* corresponds to. In case of axes X and Z, CW rotation increases the position
* (ie, moves the apparatus in a greater position in device-space coordinates),
* while the opposite holds true for axis Y.
*
* Values producing duty cycles on axes X, Y and Z:
* — Axis X:
* — Increment: @c 411
* — Decrement: @c 345
```

```
* — Axis Y:  
* — Increment: @c 350  
* — Decrement: @c 412  
* — Axis Z:  
* — Increment: @c 415  
* — Decrement: @c 365  
*/  
#define MTR_X_INC (411)  
  
/**  
* @brief Velocity setting for OCR1B to move along the negative direction on axis  
* X.  
*  
* For details, see #MTR_X_INC.  
*/  
#define MTR_X_DEC (345)  
  
/**  
* @brief Velocity setting for OCR1A to move along the positive direction on axis  
* Y.  
*  
* For details, see #MTR_X_INC.  
*/  
#define MTR_Y_INC (350)  
  
/**  
* @brief Velocity setting for OCR1A to move along the negative direction on axis  
* Y.  
*  
* For details, see #MTR_X_INC.  
*/  
#define MTR_Y_DEC (412)  
  
/**  
* @brief Velocity setting for OCR1B to move along the positive direction on axis  
* Z.  
*  
* For details, see #MTR_X_INC.  
*/  
#define MTR_Z_INC (417)  
/**  
* @brief Velocity setting for OCR1B to move along the negative direction on axis  
* Z.  
*  
* For details, see #MTR_X_INC.  
*/  
#define MTR_Z_DEC (358)
```

```
/**  
 * Velocity setting for either OCR1A or OCR1B to stop and hold current position.  
 *  
 * This corresponds to braking. For details, see #MTR_X_INC.  
 */  
#define MTR_BRAKE (380)  
  
/**  
 * @brief Converts the given amount of *grid-steps* to *pulse-steps*.  
 *  
 * Calculates the number of pulses needed to perform the given amount of  
 * transitions in device coordinate system.  
 *  
 * Also, see #GRID_X_LEN, #GRID_Y_LEN and #GRID_Z_LEN.  
 */  
#define GRID_TO_STEP(x) (x*4 - 1)  
  
/**  
 * @brief Converts the given amount of *pulse-steps* to *grid-steps*.  
 *  
 * Calculates the number of grid transitions the given amount of pulses  
 * corresponds to in device coordinate system.  
 *  
 * Also, see #GRID_X_LEN, #GRID_Y_LEN and #GRID_Z_LEN.  
 */  
#define STEP_TO_GRID(x) ((x + 1)/4)  
  
/**  
 * @brief Flag-bit of #motor_status to indicate the motors are currently  
 * resetting.  
 *  
 * Also, see #motor_status, #MTR_IS_Z, #MTR_RESET_X_DONE,  
 * #MTR_RESET_Y_DONE and #MTR_RESET_Z_DONE.  
 */  
#define MTR_RESET 0  
  
/**  
 * @brief Flag-bit of #motor_status to distinguish between motors X and Z.  
 *  
 * Motors X and Z receive their PWM signal from the same μC pin (although the  
 * signal is actually propagated to only one at a time -- see #MTR_XZ in relation  
 * to #MTR_ROUTE_X() and #MTR_ROUTE_Z()). This flag is a means to identify which  
 * motor is being operated upon.  
 *  
 * Also, see #motor_status, #MTR_RESET, #MTR_RESET_X_DONE,  
 * #MTR_RESET_Y_DONE and #MTR_RESET_Z_DONE.  
 */  
#define MTR_IS_Z 1
```

```

/**
 * @brief Flag-bit of #motor_status to indicate the position of motor X has been
 * reset.
 *
 * This flag is set by the ISR that responds to limit switch interrupts and is
 * used by #motor_reset() to determine the state of the motor reset process.
 * Motors X and Y are reset concurrently and when both their corresponding
 * flag-bits (#MTR_RESET_X_DONE and #MTR_RESET_Y_DONE) are set, the operation is
 * finalized by #motor_reset().
 *
 * Also, see #motor_status, #MTR_RESET, #MTR_IS_Z and
 * #MTR_RESET_Z_DONE.
 */
#define MTR_RESET_X_DONE 2

/**
 * @brief Flag-bit of #motor_status to indicate the position of motor Y has been
 * reset.
 *
 * This flag is set by the ISR that responds to limit switch interrupts and is
 * used by #motor_reset() to determine the state of the motor reset process.
 * Motors X and Y are reset concurrently and when both their corresponding
 * flag-bits (#MTR_RESET_X_DONE and #MTR_RESET_Y_DONE) are set, the operation is
 * finalized by #motor_reset().
 *
 * Also, see #motor_status, #MTR_RESET, #MTR_IS_Z and
 * #MTR_RESET_Z_DONE.
 */
#define MTR_RESET_Y_DONE 3

/**
 * @brief Flag-bit of #motor_status to indicate the position of motor Z has been
 * reset.
 *
 * Like #MTR_RESET_X_DONE and #MTR_RESET_Y_DONE, this flag is set by the ISR that
 * responds to limit switch interrupts and is used by #motor_reset() to determine
 * the state of the motor reset process.
 *
 * Motor Z is always the first motor to be reset independently from the other
 * two. Once this is set (ie, motor Z has been reset), #motor_reset() will
 * initiate resetting of motors X and Y.
 *
 * Also, see #motor_status, #MTR_RESET, #MTR_IS_Z,
 * #MTR_RESET_X_DONE and #MTR_RESET_Y_DONE.
 */
#define MTR_RESET_Z_DONE 4

```

```
/**  
 * @brief Flag—bit of #motor_status to indicate that no other actions has been  
 * performed since the last motor reset.  
 *  
 * It is cleared once #new_pos has been reached (ie, all steps have been  
 * performed). This flag is important in determining whether a particular  
 * position is unreachable. A position is deemed 'unreachable' and no further  
 * attempts take place when the first attempt to reach it after a fresh reset  
 * results in engaging a limit.  
 */  
#define MTR_IS_RST_FRESH 5  
  
/**  
 * @brief Flag—bit of #motor_status to indicate a limit has been engaged while  
 * under normal motor operation.  
 *  
 * This will designate to #motor_reset() that is should update the position to  
 * #new_pos after the reset cycle has been completed.  
 */  
#define MTR_LIMIT 6  
  
/**  
 * @brief A request to reposition the motors has been successfully completed.  
 *  
 * The position supplied on this event contains the current motor position. Upon  
 * invocation, the motors are quiescent and a new position may be requested.  
 */  
#define MTR_EVT_OK 1  
  
/**  
 * @brief A request to reposition the motors has been successfully completed.  
 *  
 * The position supplied on this event contains the new position that will be  
 * attempted. Upon invocation, the motors are quiescent and the current position  
 * may be requested.  
 */  
#define MTR_EVT_BUSY 2  
  
/**  
 * @brief Call motor callback if it has been set.  
 */  
#define MTR_CALL(pos, evt) \  
if(motor_callback)(*motor_callback)(pos, evt)  
  
/**  
 * @brief Initializes all pins and registers used for motor operation.  
 *  
 * Motors are operated through #motor_reset(), #motor_set() and #motor_get().
```

```

*/
void motor_init();

< /**
 * @brief Function to call on various motor-related events.
 *
 * For a list of advertised events, see the various MTR_EVT_* macros.
 *
 * @param[in] callback The callback function receives the position of the motors
 * and an event code.
 */
void motor_set_callback(void (*callback)(Position pos, uint8_t event));

< /**
 * @brief Resets the motors to a known state (homing to absolute zero).
 *
 * This function should be invoked before attempting to use the motors
 * (#motor_set() or #motor_get()). Any operation in progress at the time of
 * invocation occurs will be terminated.
 */
void motor_reset();

< /**
 * @brief Return the current device operating limits.
 *
 * @param[out] max Variable in which to return the current maximum acceptable X,
 * Y and Z coordinates.
 */
void motor_get_max(Position* max);

< /**
 * @brief Set the operating limits of the device.
 *
 * Generally, the operating limits should not exceed the physical limits of the
 * device (ie, #GRID_X_LEN, #GRID_Y_LEN and #GRID_Z_LEN). Also, they should allow
 * for at least a one-by-one X-Y grid, and at least two discrete translations on
 * Z. In any such case,
 * the limits are not altered. Successfully modifying the operating limits
 * results in a motor reset. This function may be invoked at any time; like
 * motor_reset(), any operation in progress at the time this occurs, will be
 * terminated.
 *
 * @param[in] max Variable that holds the new operating limits.
 * @returns @c 0, if the limits where acceptable; non-zero, otherwise.
 */
int8_t motor_set_max(Position* max);

/**

```

```
* @brief Move the device to the given position.  
*  
* Calling this function will result in operating the motors and will fail should  
* the motors already be operated upon or @p target lies outside the available  
* device space (#GRID_X_LEN, #GRID_Y_LEN, #GRID_Z_LEN). Also, homing should be  
* performed (by calling #motor_reset()); otherwise, the behaviour is unknown.  
*  
* This function is non-blocking (see #motor_update()).  
*  
* @param[in] target The new device position.  
* @returns @c 0, if @p target position is valid and the apparatus will be  
* configured to reach it; @c -1, otherwise.  
*/  
int8_t motor_set(Position target);  
  
/**  
* @brief Announces the current position of the device.  
*  
* This function will fail (return @c -1) if the motors are currently resetting  
* or otherwise operated upon.  
*  
* @param[out] pos The value of #cur_pos.  
* @returns @c 0, if @p pos was set; @c -1, otherwise.  
*/  
int8_t motor_get(Position *pos);  
  
/**  
* @brief Activates the appropriate motors in order to reach #new_pos.  
*  
* It determines which motors should be operated and for how many steps. Motors X  
* and Y may be set-up to operate concurrently but only for the same distance  
* (irrespective of direction); once the common part has been completed, a new  
* invocation of this function will configure the other motor to perform the  
* remainder of steps. Note that #cur_pos should be updated before this new  
* invocation occurs. Motor Z takes precedence over motors X and Y when its new  
* position (@link #new_pos new_pos.z@endlink) is greater than its current  
* (@link #cur_pos cur_pos.z@endlink) (that is the  
* sensor head will first be retracted and then moved on X-Y plane). The reverse  
* holds true when lowering the head.  
*  
* This function should only be called while all motors are at rest  
* (#PWM_IS_ON()). Generally, this occurs from within #motor_set(), which will  
* make sure no operation is already in process, and from the ISR that responds  
* to the completion of the specified amount of steps (@c TIMERO_COMPA_vect).  
* Typically, 3 to 4 invocations suffice; a return value of @c -1 designates that  
* #cur_pos has reached #new_pos and no set-up has been performed. In this case,  
* all the unnecessary circuitry (Timer/Counters and rotary encoder) should be  
* disabled by the callee.
```

```

*
* This function is non-blocking and is not safe to call while the motors are
* being operated upon.
*
* @returns @c 0 upon activating the motors; @c -1 otherwise.
*/
static int8_t motor_update();

/**
* @brief Request the AutoLock to be enabled for the specified amount of steps.
*
* This function initiates the step counter, ie, any pulses sensed on pin #MUX_2Z
* after calling this function will increment the step counter. Once the amount
* of steps specified by @p steps has been reached, #MTR_nLOCK will be enabled by
* the underline hardware blocking the transmission of the PWM signal on #MTR_XZ
* and #MTR_Y. Also, an Output Compare Match Interrupt will trigger (for vector
* @c TIMERO_COMPA_vect) which, ultimately, completely disables the
* Timer/Counters.
*
* The AutoLock mechanism utilizes Timer/Counter0 for counting the steps and is,
* thus, guaranteed that only the requested amount of steps will be performed,
* even if the MCU is busy attending to other tasks when this occurs. Once the
* MCU is available to execute the aforementioned ISR, it will either forward
* another request or completely disable the Timer/Counters.
*
* @param[in] steps The amount of steps to count before the AutoLock is enabled.
* An additional transition is always performed, ie passing in @c 1 will
* trigger AutoLock after two steps and so on. This is due to "Compare Match
* Blocking by TCNT0 Write" feature as described by *Atmel p97*. As a special
* case, passing in @c 0 allows a maximum of 256 steps to be performed.
*/
static void setup_lock(uint8_t steps);

/**
* @brief Prepares motion on the specified axis and direction.
*
* This configures PWM generation and propagation to the specified motor and the
* settings for an appropriate velocity and rotary encoder feedback. Calling this
* function is generally followed by a call to #motor_start().
*
* Because #MUX_S0 and #MUX_S1 select between motors X and Z apart from which
* rotary encoder to activate, if concurrent motion is required (for instance, on
* both axes X and Y), axis Y should be the *first* one to set-up to avoid
* disabling propagation to the other. In this configuration, it is the rotary
* encoder of the *last* axis set-up that will provide feedback. Note that
* #MotorAxis values may not be OR-ed together!
*
* Motors X and Z share the same signal line (on pin @c OC1A) and, thus, may not

```

```

* be operated at the same time.

*
* @param[in] axis The axis to set-up.
* @param[in] dir The direction of motion.
*/
static void setup_axis(MotorAxis axis, MotorDir dir);

/**
* @brief Wrapper around #MTR_PWM_START().
*
* This function is provided as a complement to #motor_stop(). As in the case of
* #MTR_PWM_START(), which is what actually enables PWM generation, this function
* should be called *after* applying the desired velocity settings using
* #setup_axis().
*/
static void motor_start();

/**
* @brief Disables the step counter, the PWM timer and the rotary encoder.
*/
static void motor_stop();

/**
* @brief Backtrack a motor that has engaged a limit switch.
*
* In order for this to operate correctly, @c OCR1A/B and #MTR_IS_Z, if needed,
* must be populated with an appropriate value beforehand. This means that this
* function cannot be used to backtrack on an axis the limit switch of which has
* been engaged *before* the initiation of motion (such as before device
* power-on).
*
* Note that the position of the backtracked motor in #cur_pos is not updated.
* This function should be followed by another operation to update the position
* (eg, reset).
*
* @returns The axis that has been backtracked; @c 0 in the event that no limit
* switch signal read as logic low.
*/
static MotorAxis motor_backtrack();

#endif /* MOTOR_H_INCL */
/** @} */

```

**motor.c**

```

#include "motor.h"

#include <avr/io.h>

```

```

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

/***
* @ingroup motor
* @brief Contains flags concerning the current status of the motors.
*
* See #MTR_IS_Z, #MTR_RESET, #MTR_RESET_X_DONE, #MTR_RESET_Y_DONE
* and #MTR_RESET_Z_DONE.
*/
static uint8_t motor_status = 0;

/***
* @ingroup motor
* @brief The current (absolute) position of the device apparatus.
*
* #motor_reset() should be invoked to initialize the state of both the
* (physical) device and of this variable.
*
* To move the apparatus to a new position, #motor_set() should be invoked. Also,
* see #new_pos.
*/
static Position cur_pos;

/***
* @ingroup motor
* @brief The new position the device will attempt to get to.
*
* It is updated via #motor_set() which will also activate the motors.
*/
static Position new_pos;

/***
* @ingroup motor
* @brief Pointer to function to call on various motor events.
*/
static void (*motor_callback)(Position pos, uint8_t status);

/***
* @ingroup motor
* @brief Operating (maximum) limits of the device.
*
* The operating limits are managed via motor_get_max() and motor_set_max() and
* may be set to anything less than or equal to #GRID_X_LEN, #GRID_Y_LEN and
* #GRID_Z_LEN.
*/
static Position max_pos = { .x = GRID_X_LEN, .y = GRID_Y_LEN, .z = GRID_Z_LEN };

```

```
void motor_init() {

    /* Set backtrack control line as output. */
    BCK_Y_DDR |= _BV(BCK_Y);
    BCK_XZ_DDR |= _BV(BCK_XZ);

    /* Backtrack MOSFETs should be off under normal conditions. */
    BCK_Y_PORT &= ~_BV(BCK_Y);
    BCK_XZ_PORT &= ~_BV(BCK_XZ);

    MTR_nLOCK_DDR |= _BV(MTR_nLOCK); /* Set motor AutoLock pin as output. */

    /* Set motor signal pins as output. */
    MTR_Y_DDR |= _BV(MTR_Y);
    MTR_XZ_DDR |= _BV(MTR_XZ);

    /* Trigger an interrupt whenever the specified amount of steps has been
     * completed. Steps are compared against @c OCROA. */
    TIMSK0 |= _BV(OCIEOA);

    /* Set signal from the limit switches as input. */
    LMT_nXZ_DDR &= ~_BV(LMT_nXZ);
    LMT_nY_DDR &= ~_BV(LMT_nY);

    /* Set the value that produces a 50Hz PWM frequency. */
    ICR1 = MTR_TOP;

    /* Operate PFCPWM with @c TOP value being set in @c ICR1 (WGM13:0 = 8). */
    TCCR1B = _BV(WGM13);

    /* Enable pin change interrupts on pins connected to the limit switches. */
    PCICR |= _BV(LMT_PCIE);
    LMT_PCMSK |= LMT_PCMSK_VAL;
}

void motor_set_callback(void (*callback)(Position pos, uint8_t event)) {
    motor_callback = callback;
}

void motor_reset() {

    /* Begin with resetting axis Z if no reset is already in progress. It is
     * imperative to first retract on axis Z separately from the others. Once
     * that has been dealt with, axes X and Y may follow. */
    if(bit_is_clear(motor_status, MTR_RESET)) {
        motor_stop();
        motor_status |= _BV(MTR_RESET) | _BV(MTR_IS_Z);
        setup_axis(AXIS_Z, MTR_INC);
    }
}
```

```

LOCK_DISABLE();
motor_start();

/* Axis Z has been reset. Now, axes X and Y may follow. */
} else if(bit_is_set(motor_status, MTR_RESET_Z_DONE)) {
    /* Remove flag denoting axis Z is resetting. */
    motor_status &= ~(_BV(MTR_IS_Z) | _BV(MTR_RESET_Z_DONE));

    /* Reset axes X and Y. */
    setup_axis(AXIS_Y, MTR_DEC);
    setup_axis(AXIS_X, MTR_DEC);
    LOCK_DISABLE(); /* Manually enable PWM propagation. */
    motor_start();

    /* All resetting stages have been completed. Reset #cur_pos and flags. */
} else if(bit_is_set(motor_status, MTR_RESET_X_DONE)
        && bit_is_set(motor_status, MTR_RESET_Y_DONE)) {

    cur_pos.x = 0;
    cur_pos.y = 0;
    cur_pos.z = max_pos.z - 1;
    motor_stop();

    LOCK_ENABLE();

    /* If this resetting cycle was initiated as a response to a limit being
     * engaged while under normal motor operation, retry reaching #new_pos
     * anew. */
    if(bit_is_set(motor_status, MTR_LIMIT)) {
        if(motor_update()) {
            motor_stop();
            MTR_CALL(cur_pos, MTR_EVT_OK);
        }
    } else {
        new_pos = cur_pos;
        MTR_CALL(cur_pos, MTR_EVT_OK);
    }

    motor_status &= ~(_BV(MTR_RESET) | _BV(MTR_LIMIT)
                    | _BV(MTR_RESET_X_DONE) | _BV(MTR_RESET_Y_DONE));
    motor_status |= _BV(MTR_IS_RST_FRESH);

    /* Reset is in progress. */
} else {
}
}

void motor_get_max(Position* max) {

```

```
max->x = max_pos.x;
max->y = max_pos.y;
max->z = max_pos.z;
}

int8_t motor_set_max(Position* max) {
    if(max->x > GRID_X_LEN || max->y > GRID_Y_LEN || max->z > GRID_Z_LEN
    || max->x < 1 || max->y < 1 || max->z < 2) {
        return -1;
    }
    max_pos.x = max->x;
    max_pos.y = max->y;
    max_pos.z = max->z;
    motor_reset();
    return 0;
}

int8_t motor_set(Position target) {

    /* Fail, if the motors are resetting or otherwise operated upon. */
    if(bit_is_set(motor_status, MTR_RESET) || PWM_IS_ON()) return -1;

    /* Fail, if target coordinates lay outside the available device space. */
    if(target.x >= max_pos.x
    || target.y >= max_pos.y
    || target.z >= max_pos.z) {
        return -1;
    }

    new_pos = target;

    return motor_update();
}

int8_t motor_get(Position *pos) {
    if(bit_is_set(motor_status, MTR_RESET) || PWM_IS_ON()) return -1;
    pos->x = cur_pos.x;
    pos->y = cur_pos.y;
    pos->z = cur_pos.z;
    return 0;
}

static int8_t motor_update() {
    uint8_t steps = 0;

    /* Motion along axes X and Y takes precedence over motion along axis Z, when
     * no submersion is requested on the latter. Also, only motion along axes X
     * and Y may be combined with one another. */
}
```

```

if((new_pos.x != cur_pos.x || new_pos.y != cur_pos.y)
&& new_pos.z <= cur_pos.z) {

    /* Relative offset from #cur_pos on axes X and Y. */
    int16_t rel_x = (int16_t)new_pos.x - (int16_t)cur_pos.x;
    int16_t rel_y = (int16_t)new_pos.y - (int16_t)cur_pos.y;

    /* Enable lines for PWM propagation to motor Y and set its speed. */
    if(rel_y) {
        setup_axis(AXIS_Y, rel_y); /* @c rel_y is used only for its sign. */
    }

    /* Enable lines for PWM propagation to motor X and set its speed. Motor
     * X may only be set-up after motor Y, as noted in #setup_axis(). */
    if(rel_x) {
        setup_axis(AXIS_X, rel_x); /* @c rel_x is used only for its sign. */
    }

    rel_x = abs(rel_x);
    rel_y = abs(rel_y);

    if(rel_x > 0 && rel_y > 0) {
        /* Request the common amount of steps between @c rel_x and
         * @c rel_y. */
        steps = rel_x >= rel_y ? GRID_TO_STEP(rel_y)
                               : GRID_TO_STEP(rel_x);

    } else {
        /* If one of @c rel_x and @c rel_y is @c 0, then calculate the
         * amount of steps solely based on the other (non-zero) value. */
        steps = rel_x > 0 ? GRID_TO_STEP(rel_x)
                           : GRID_TO_STEP(rel_y);
    }

    /* Configure motion along axis Z. */
} else if(new_pos.z != cur_pos.z) {

    int16_t rel_z = (int16_t)new_pos.z - cur_pos.z;
    setup_axis(AXIS_Z, rel_z); /* @c rel_z is used only for its sign. */
    steps = GRID_TO_STEP(abs(rel_z));
    motor_status |= _BV(MTR_IS_Z);

} else {
    /* Already at #new_pos. */
    motor_status &= ~_BV(MTR_IS_Z);
    return -1;
}

```

```
setup_lock(steps);
motor_start();
return 0;
}

static void setup_lock(uint8_t steps) {

    /* Clear any previous steps. */
    TCNTO = 0; /* Isn't this done automatically? */

    /* Number of steps to count. A transition from white to black counts as one
     * step. */
    OCROA = steps;

    /* Toggle @c OCOA pin (ie, #MTR_nLOCK) on compare-match from high to low.
     * This way, the PWM signal is no longer propagated to the motors after the
     * specified amount of steps has been performed (AutoLock mechanism). In
     * order to generate output on @c OCOA (or even, produce an interrupt) based
     * on a *custom* value, CTC mode should be used instead of Normal mode (which
     * always counts up to @c MAX). */
    TCCROA |= _BV(COMOAO) | _BV(WGM01);

    /* Use @c T0 pin as a clock source. @c T0 is connected to the (multiplexed)
     * @c STEP output from the optical encoders. The counter is configured to
     * increment on every falling edge, ie, on a transition from a white stripe
     * of the encoder belt to a black one. Force an initial compare match to set
     * @c OCOA and disable the lock. */
    TCCROB |= _BV(FOCOA) | _BV(CS02) | _BV(CS01);

    /* Due to unknown reasons, when the lock (#MTR_nLOCK) is manually operated
     * (while Timer/Counter0 is, of course, disabled), there seems to be a false
     * outcome of @c FOCOA. In particular, though @c OCOA is initially set,
     * after a few cycles, it returns to @c 0. The small delay below is enough to
     * to determine whether the forced compare match has rendered a logic high.
     * If not, it is forced again. */
    _delay_us(100);
    if(bit_is_clear(MTR_nLOCK_PIN, MTR_nLOCK)) TCCROB |= _BV(FOCOA);
}

static void setup_axis(MotorAxis axis, MotorDir dir) {
    uint16_t speed;
    uint8_t is_inc = dir >= MTR_INC;

    MUX_DISABLE();
    switch(axis) {
        case AXIS_Y:
            speed = is_inc ? MTR_Y_INC : MTR_Y_DEC;
```

```

/* Axis Y uses OCR1A for PWM generation. */
OCR1A = speed; /* Set angular velocity. */
MTR_ROUTE_Y();
MUX_ENABLE(); /* Enable rotary encoder. */
PWM_Y_ENABLE(); /* Override port operation for PWM waveform. */
break;

case AXIS_X:
case AXIS_Z:

/* Propagate PWM signal to either axis X or Z. */
if(axis == AXIS_Z) {
    speed = is_inc ? MTR_Z_INC : MTR_Z_DEC;
    MTR_ROUTE_Z();
} else {
    speed = is_inc ? MTR_X_INC : MTR_X_DEC;
    MTR_ROUTE_X();
}

/* Axes X and Z use OCR1B for PWM generation. */
OCR1B = speed;
MUX_ENABLE(); /* Enable rotary encoder. */
PWM_XZ_ENABLE(); /* Override port operation for PWM signal. */
break;
}

static void motor_start() {
    MTR_CALL(new_pos, MTR_EVT_BUSY);

    /* This is what actually enables PWM generation and should be called after
     * preparing the Timer/Counters (velocity settings). */
    MTR_PWM_START();
}

static void motor_stop() {

    /* Stop motor PWM generation (Clock Select bits set to @c 0). */
    MTR_PWM_STOP();

    /* Stop step counter. */
    TCCR0B &= ~(_BV(CS02) | _BV(CS01) | _BV(CS00));

    /* Release ports from PWM operation. */
    PWM_Y_DISABLE();
    PWM_XZ_DISABLE();

    /* Deactivate rotary encoders. */
}

```

```
MUX_DISABLE();

/* Reset PWM duty cycle settings. */
OCR1A = 0;
OCR1B = 0;

/* Release Lock pin from @c OC0A so it may be operated by software, if and
 * when needed. */
TCCROA &= ~(_BV(COM0AO) | _BV(COM0A1));

LOCK_ENABLE();
}

static MotorAxis motor_backtrack() {
    MotorAxis axis; /* _BV() of one of #MTR_RESET_[X|Y|Z]_DONE. */

    /* Remove Clock Select to stop counter and make updates in @c OCR1A/B. */
    MTR_PWM_STOP();

    /* Limit X or Z detected. Axes X and Z limit switches share the same
     * circuitry; identify which one has reached its limit. */
    if(IS_LMT_nXZ()) {

        /* Reverse the (direction) of the angular velocity. */
        if(bit_is_set(motor_status, MTR_IS_Z)) {
            OCR1B = OCR1B == MTR_Z_INC ? MTR_Z_DEC : MTR_Z_INC;
            axis = AXIS_Z;

        } else {

            OCR1B = OCR1B == MTR_X_INC ? MTR_X_DEC : MTR_X_INC;
            axis = AXIS_X;
        }
        BCK_XZ_PORT |= _BV(BCK_XZ);
        MTR_PWM_START();

        /* Wait until the limit switch is disengaged (active low pin). */
        loop_until_bit_is_set(LMT_nXZ_PIN, LMT_nXZ);
        /* Then wait for a logic low from the rotary encoder (black stripe). */
        loop_until_bit_is_clear(MUX_2Z_PIN, MUX_2Z);

        MTR_PWM_STOP();
        BCK_XZ_PORT &= ~_BV(BCK_XZ);
        OCR1B = MTR_BRAKE;
        MTR_PWM_START();

        /* Limit Y detected. */
    } else if(IS_LMT_nY()) {
```

```

OCR1A = OCR1A == MTR_Y_INC ? MTR_Y_DEC : MTR_Y_INC;
axis = AXIS_Y;

BCK_Y_PORT |= _BV(BCK_Y);
MTR_PWM_START();

loop_until_bit_is_set(LMT_nY_PIN, LMT_nY);

/* Motors X and Y reset at the same time while feedback is available
 * from the rotary encoder for X (#MTR_ROUTE_X()). Now that backtracking
 * is required on axis Y, it should receive feedback from its appropriate
 * encoder. */
MTR_ROUTE_Y();
loop_until_bit_is_clear(MUX_2Z_PIN, MUX_2Z);

MTR_PWM_STOP();

/* Reinstate feedback from motor X rotary encoder. Even if motor X has
 * already been reset, #motor_reset() is responsible for deactivating all
 * necessary circuitry, including that of MTR_ROUTE_X(). */
MTR_ROUTE_X();
BCK_Y_PORT &= ~_BV(BCK_Y);
OCR1A = MTR_BRAKE;
MTR_PWM_START();

} else {
    axis = 0;
}

return axis;
}

/**
 * @ingroup motor
 * @brief Responds to the completion of the specified amount of steps.
 *
 * It is responsible to update #cur_pos based on the amount of the steps
 * performed. #motor_update() is used to determine whether further motor
 * operation is required to reach the position specified by #new_pos. If not, the
 * motor circuitry is deactivated.
 */
ISR(TIMERO0_COMPA_vect) {
    uint8_t offset; /* The number of grid-steps performed. */

    MTR_PWM_STOP(); /* Stop PWM generation. */
    PWM_XZ_DISABLE(); /* Release signal pins from Output Compare Registers. */
    PWM_Y_DISABLE();
}

```

```
/* Determine the amount of transitions on the grid. This is the relative
 * offset from #cur_pos not taking into consideration the direction of
 * motion. */
offset = STEP_TO_GRID(OCROA);

/* If @c OCR1A is set, a PWM signal was generated and propagated to motor Y.
 * Update #cur_pos.y by the amount of steps performed. */
if(OCR1A) {
    /* Alter #cur_pos.y @c offset in the appropriate direction. */
    cur_pos.y += OCR1A == MTR_Y_INC ? offset : -offset;

    /* Remove any settings of PWM generation. */
    OCR1A = 0;
}

/* If @c OCR1B is set, a PWM signal was generated and propagated to either
 * motor X or Z. Update #cur_pos.x or #cur_pos.z by the amount of steps
 * performed after determining which motor (X or Z) was being operated. */
if(OCR1B) {

    if(bit_is_set(motor_status, MTR_IS_Z)) {
        motor_status &= ~_BV(MTR_IS_Z);
        cur_pos.z += OCR1B == MTR_Z_INC ? offset : -offset;

    } else {
        cur_pos.x += OCR1B == MTR_X_INC ? offset : -offset;
    }

    /* Remove any settings of PWM generation. */
    OCR1B = 0;
}

/* Ensure there are no more steps to perform. If there are not any,
 * completely disable the motor circuits. */
if(motor_update()) {
    motor_stop();

    /* The motors have successfully reached their destination. Remove
     * #MTR_FRESH_RST flag as they are not reset any more. */
    motor_status &= ~_BV(MTR_IS_RST_FRESH);

    /* Inform that motors have reached their destination. */
    MTR_CALL(cur_pos, MTR_EVT_OK);
}

/**
```

```

* @ingroup motor
* @brief Responds to limit switch interrupts.
*
* If a limit is engaged under normal motor operation (ie, while attempting to
* reach #new_pos), it means the device has failed to properly track its state.
* A #motor_reset() is forced in this case. Later, the device is configured to
* reach #new_pos anew.
*
* This ISR also heavily affects the motor resetting cycle (#motor_reset()) by
* setting flags #MTR_RESET_X_DONE, #MTR_RESET_Y_DONE and #MTR_RESET_Z_DONE.
*/
ISR(PCINT1_vect) {

    /* Force a small delay during which the signal stabilizes. */
    _delay_ms(50);

    /* This ISR is executed whenever there is a Pin Change, that is from @c 1 to
     * 0 *and* vice versa! In the event of a pin settling back to @c 1 (idle)
     * after a switch has been disengaged, simply ignore it. */
    if(!IS_LMT_nXZ() && !IS_LMT_nY()) return;

    /* Deactivate step counter for it is not needed while backtracking or
     * resetting. */
    TCCROA &= ~(_BV(COM0AO) | _BV(COM0A1) | _BV(WGM01));
    LOCK_DISABLE();

    MotorAxis axis = motor_backtrack();

    /* Limit while in motor reset. */
    if(bit_is_set(motor_status, MTR_RESET)) {

        switch(axis) {
            case AXIS_X:
                motor_status |= _BV(MTR_RESET_X_DONE);
                break;
            case AXIS_Y:
                motor_status |= _BV(MTR_RESET_Y_DONE);
                break;
            case AXIS_Z:
                motor_status |= _BV(MTR_RESET_Z_DONE);
                break;
            default:
                /* Backtrack error / no axis. N/A; could be omitted. */
                return;
        }

        /* Limits have been engaged after two successive resets. */
    } else if(bit_is_set(motor_status, MTR_IS_RST_FRESH)) {

```

```

/* Special case: If an unexpected limit occurs right after resetting, it
 * means that position is unreachable (probably because somebody has
 * altered the physical limits). Dealing with this also means the device
 * will not fall into an infinite loop should an insurmountable obstacle
 * happen in its path. */

/* Maybe it would be of interest to redefine the device-space dimensions
 * based on the actual current working space. Otherwise, no particular
 * action should be taken, other than *not* setting #MTR_LIMIT (which
 * indicates that getting to #new_pos should be attempted again after a
 * motor reset. */

/* Destination unreachable. */
motor_stop();

/* Limit engaged while under normal motor operation. */
} else {
    /* Unexpected limit. */
    motor_status |= _BV(MTR_LIMIT);
    motor_stop();
}
motor_reset();
}

```

## W5100 Socket handler

### net.h

```

/**
 * @file
 */

#ifndef NET_H_INCL
#define NET_H_INCL

#include <inttypes.h>

/**
 * @brief Responsible for dealing with interrupts on #HTTP_SOCKET.
 *
 * It handles the various TCP states and calls srvr_call() when data are
 * available. It is also responsible to reopen the socket, once a connection has
 * been terminated.
 *
 * @param[in] s This Socket (@c 0--@c 3).
 * @param[in] status The #NET_Sn_IR value at the time of invocation.
 */
void handle_http_socket(uint8_t s, uint8_t status);

```

```

#endif /* NET_H_INCL */

net.c

#include 'net.h'
#include 'defs.h'
#include 'w5100.h'
#include 'http_server.h'

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

/** 
 * @brief Propagates interrupts from the W5100 to the appropriate handlers.
 *
 * This ISR reads the #NET_IR to determine the socket source and calls the
 * appropriate handler. Currently, only one socket operates, which is responsible
 * for the HTTP server transactions (see handle_http_socket()). Upon invoking the
 * handler, the ISR keeps a copy of the socket's interrupt register (#NET_Sn_IR)
 * and passes its value as an argument. The handler is supposed to respond to the
 * specified flags only. Once completed, the ISR will clear the interrupt flag
 * bits that were specified to the handler. If more interrupt flag bits have been
 * set in the meantime, the ISR will be ready to respond to them.
 */
ISR(INT1_vect) {
    uint8_t status = net_read8(NET_IR);
    uint8_t socket;

    /* Only HTTP_SOCKET interrupts are of interest; clear high nibble. */
    net_write8(NET_IR, 0xE0);

    if(bit_is_set(status, HTTP_SOCKET)) {
        socket = net_read8(NET_Sn_IR(HTTP_SOCKET));
        handle_http_socket(HTTP_SOCKET, socket);

        /* Clear Socket interrupt flags that have just been dealt with. */
        net_write8(NET_Sn_IR(HTTP_SOCKET), socket);
    }
}

void handle_http_socket(uint8_t s, uint8_t status) {

    /* Data available. */
    if(bit_is_set(status, NET_Sn_IR_RECV)) {

        if(net_read16(NET_Sn_RX_RSR(s)) > 0 ) {
            uint8_t c; /* Discarded character. */
            int8_t c_type;
    }
}

```

```

/* Stream data from this Socket to local (host) buffering. */
set_socket_buf(s);

/* Service HTTP request. */
srvr_call();

/* Discard the remainder of the request. */
while((c_type = s_next(&c)) != -1) {
}

}

/* Occurs when a connection termination is requested OR completed. */
if(bit_is_set(status, NET_Sn_IR_DISCON)) {
    uint8_t sn_SR = net_read8(NET_Sn_SR(s));

    /* Termination request is received from peer host. Close the connection
     * by sending a DISCON or CLOSE command. *WIZnet p.30* */
    if(sn_SR == NET_Sn_SR_CLOSEWAIT || sn_SR == NET_Sn_SR_CLOSED) {
        /* Issue a DISCON command as a response to FIN+ACK. */
        net_write8(NET_Sn_CR(s), NET_Sn_CR_DISCON);
        net_write8(NET_Sn_CR(s), NET_Sn_CR_CLOSE);

        /* Re-open socket. */
        net_socket_open(HTTP_SOCKET, NET_Sn_MR_TCP, HTTP_PORT);
        net_write8(NET_Sn_CR(s), NET_Sn_CR_LISTEN);
    }
}

if(bit_is_set(status, NET_Sn_IR_TIMEOUT)) {
    net_write8(NET_Sn_CR(s), NET_Sn_CR_DISCON);
}
}

```

## Πρωτόκολλο 1-Wire

### onewire.h

```

/**
 * @file
 * @addtogroup 1_wire 1-wire bus
 * @{
 *
 * @brief Single drop 1-wire bus interface.
 *
 * The current implementation provides functions to reset, read and write the DQ
 * line. No device negotiation is implemented (ROM Search).
 */

```

```
#ifndef ONEWIRE_H_INCL
#define ONEWIRE_H_INCL

#include "defs.h"

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

#include <inttypes.h>

/***
 * @brief ROM command: access memory functions without supplying device code.
 */
#define W1_ROM_SKIP 0xCC

/***
 * @brief Memory command: start reading scratchpad memory contents.
 */
#define W1_READ_SCRATCHPAD 0xBE

/***
 * @brief Memory command: begin temperature conversion.
 */
#define W1_CONVERT_T 0x44

/***
 * @brief The bit read over 1-wire DQ line.
 *
 * An appropriate read slot should be initiated before reading this value.
 *
 * @returns 0, if the slave transmitted a 0; non-zero, otherwise.
 */
#define W1_READ() \
(W1_DQ_PIN & _BV(W1_DQ))

/***
 * @brief Pull 1-wire DQ low.
 */
#define W1_DQ_LOW() \
W1_DQ_PORT &= ~_BV(W1_DQ); \
W1_DQ_DDR |= _BV(W1_DQ)

/***
 * @brief Release 1-wire so it may rise to high via the external resistor.
 */
#define W1_DQ_RELEASE() \
W1_DQ_DDR &= ~_BV(W1_DQ)
```

```

/** 
 * @brief Initialise the 1-wire DQ bus (reset and presence pulses).
 *
 * Upon completion, the DQ bus is ready to be operated upon immediately.
 *
 * @returns @c 0, if a presence pulse was detected; non-zero, otherwise.
 */
uint8_t w1_reset();

/** 
 * @brief Send a data byte over the 1-wire DQ line.
 *
 * Upon completion, the DQ bus is ready to be operated upon immediately.
 *
 * @param[in] byte Data to send.
 */
void w1_write(uint8_t byte);

/** 
 * @brief Read up to 16 bits from the 1-wire DQ line.
 *
 * @param[in] bits Amount of bits to read (up to 16).
 * @return Data read from DQ line.
 */
uint16_t w1_read(uint8_t bits);

#endif /* ONEWIRE_H_INCL */
/** @} */

```

**onewire.c**

```

#include "onewire.h"

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

uint8_t w1_reset() {
    uint8_t presence;

    /* Note: it is safe to write 0 on an input pin. In contrast, 1 would enable
     * the internal pull-up resistor. */

    /* The bus master transmits a reset pulse (a low signal for μ480---960s).
     * *DS18B20 p.14* */
    W1_DQ_LOW();
    _delay_us(500);

    /* The bus master then goes into Rx mode. *DS18B20 p.14* */
    W1_DQ_RELEASE();

```

```

/* The DS18B20 waits μ15—60s and then transmits the presence pulse (a low
 * signal for μ60—240s). *DS18B20 p.14*. Sample the line after μ70s,
 * during which the presence should have been set. */
_delay_us(70);

presence = W1_READ();

/* Wait for a minimum of μ480s in Master Rx mode. *DS18B20 p.18*. μ70s have
 * already been spent. Explicitly leaving a total of μ500s. */
_delay_us(430);

return presence;
}

void w1_write(uint8_t byte) {
    uint8_t i = 8;

    /* Data are transmitted LSB first (*DS18B20 p.5*). 8 iterations are
     * performed, at the end of each, @p byte is shifted to the right by one
     * bit. */
    while(i--) {

        W1_DQ_LOW();

        /* DQ must remain low for a minimum of 1 μs (*DS18B20 p.19). Then, if
         * writing @c 1, it should go high. Otherwise, it should remain low for
         * the remainder of the timeslot. */
        _delay_us(2);

        /* If writing @c 0, do not change the data-direction on the DQ pin.
         * Otherwise, do. In either case, wait for the remainder of the slot. */
        W1_DQ_DDR &= ~((byte & 0x1)*_BV(W1_DQ));

        /* Slave sampling occurs within 30 to μ60s after DQ line goes low
         * (*DS18B20 p.19*). */
        _delay_us(55);

        /* Also, allow a minimum μ1s recovery between write slots (*DS18B20
         * p.19*). Explicitly leaving a total of μ61s. */
        W1_DQ_RELEASE();

        _delay_us(2);
        byte >>= 1;
    }
}

uint16_t w1_read(uint8_t bits) {

```

```

uint16_t data = 0; /* The output. */
uint16_t mask = 0x01; /* Shifted left-wise, applied on @c data. */

while(bits--) {
    /* Generate a Read time slot */

    /* A read slot is initiated with a low pulse of  $\mu$ 1s minimum
     * (*DS18B20 p19*) */
    W1_DQ_LOW();
    _delay_us(2);

    /* Release DQ line so the slave may gain control of it. */
    W1_DQ_RELEASE();

    /* Output data is valid for  $\mu$ 15s after the falling edge of the read time
     * slot (*DS18B20 p.19*). Read near the end of the  $\mu$ 15s window. */
    _delay_us(10);

    /* Append 1 to the buffer, if necessary. */
    if(W1_READ()) {
        data |= mask;
    }

    /* Allow for a slot of at least  $\mu$ 60s  $\mu$ +1s for recovery time (*DS18B20
     * p.19*). Explicitly leaving a total of  $\mu$ 61s. */
    _delay_us(49);

    mask <= 1;
}

return data;
}

```

## Διεπαφή παραμέτρων

### param.h

```

/**
 * @file
 * @brief Link between resource handlers and stream parsers.
 *
 * Resource handlers are triggered to serve a request made on a particular
 * endpoint (or absolute path). Some of those handlers may require additional
 * input in order to perform their intended tasks. That input should be present
 * in the request itself but, in all probability, not in a format readily
 * processable by the handler but, more likely, in a serialized format that
 * facilitates data interchange and interoperability.
 *
 * That serialized input needs to be converted into something more familiar to

```

\* the nature of the handler (such as variables of integers and strings) by  
\* modules specifically designed to parse and analyse the serialized input of a  
\* particular format; the parsers.  
\* On one end, the parsers – familiar with the structure and semantics of a  
\* data format – can gain access to the values that are direly needed by the  
\* handlers on the other end, while both entities are housed within an  
\* environment of limited resources.  
\*  
\* The current approach is for the handlers to prepare the variables to be  
\* populated with values from the serialized input as well as a set of  
\* descriptions to inform the parsers what they should look for. The descriptions  
\* include:  
\* – A parameter–token (or simply, token), which is the name of a variable  
\* as it is expected to appear in the serialized input.  
\* – A data type, which would allow the parser to delegate the actual value  
\* processing to external utilities (which could be shared among all  
\* parsers and other components).  
\* – A storage location where a parsed value of a parameter should be stored.  
\* Typically, this should be the address of a variable within the scope of  
\* the handler (to avoid the need for a parser to dynamically allocate  
\* memory and return that to the handler).  
\* – A means to communicate additional settings (such as the maximum allowable  
\* data size that may be stored in the provided storage location) as well  
\* as the outcome of the operation for each parameter–token separately (for  
\* example, parameter "X" has been assigned a valid value, whereas there  
\* was no sign of parameter "Y").  
\*  
\* These are achieved by the `Cc params` module (this module). For starters, it  
\* provides the `ParamValue` data structure that may be used to specify some of the  
\* above requirements for a parameter. A variable of this type should be created  
\* for every parameter–token that is to be identified on the stream. It contains  
\* members to specify the type (`#ParamValue::type`), the storage location  
\* (`#ParamValue::data_ptr`) and the additional settings and outcome  
\* (`#ParamValue::status_len`) (the last two intermixed in one byte to conserve  
\* space). For a complete description, see `#ParamValue`.  
\*  
\* The first requirement stated in the list above, though, (the parameter–token)  
\* is not handled by this structure. The reason behind this is the knowledge of  
\* the underlying API that is used to identify character sequences from a stream  
\* (basically, the network module). That API needs to be provided a contiguous  
\* block of tokens (array of strings). If the parameter–token was specified in  
\* `#ParamValue`, it would instantly render it incompatible with that API. So,  
\* alternatively, a second wrapper structure is defined, `#ParamInfo`, one that  
\* encompasses the array of strings (`#ParamInfo::tokens`) and the array of  
\* `#ParamValue` (`#ParamInfo::values`), the two being related on a one–to–one basis.  
\* Lastly, there is an additional member that specifies the size of the two  
\* arrays (`#ParamInfo::len`).  
\*/

```
#ifndef PARAM_H_INCL
#define PARAM_H_INCL

#include <inttypes.h>

/** 
 * @brief Parameter value data types.
 *
 * Mostly needed to specify the data type for a particular parameter (to load
 * the appropriate value parser).
 *
 * Currently, only unsigned integers and strings are specified.
 */
typedef enum {
    /**
     * @brief Unsigned integer (typically, 8- or 16-bit).
     *
     * The size of the integer could be greater than 16 bits. It depends on the
     * requirements of the application and the availability of appropriate
     * parsers.
     */
    DTTYPE_UINT,
    /**
     * @brief Null-terminated character sequence. */
    DTTYPE_STRING
} DataType;

/** 
 * @brief Controls parsing for the value of a particular parameter-token.
 */
typedef struct {
    /**
     * @brief Data type for the value of this token. One of #DataType. */
    DataType type;

    /**
     * @brief Memory location where a valid value should be placed at.
     *
     * This is set by the caller and updated by the parser. That is, the caller
     * sets the address, perhaps one of its local variables, and the parser
     * functions update its contents to match the value parsed from the stream.
     */
    void* data_ptr;

    /**
     * @brief Specifies the length and the status of a parameter.
     *
     * The meaning of this member is two-fold. The two most significant bits
     */
}
```

```

* determine the status of @link ParamValue::data_ptr data_ptr@endlink, while
* the rest, the allowable size for a parsed value.
*
* The status bits (bits @c 7 and @c 6) indicate the outcome for this
* particular parameter, such as whether a valid value (#PARAM_VALID), an
* inappropriate character (#PARAM_INVALID) or too many characters
* (#PARAM_TOO_LONG) have been discovered on the stream. Their value should
* be reset to @c 00 (#PARAM_NOT_SET) *before* any parsing takes place.
* Setting these bits is, typically, performed by the assigned parser.
*
* The size bits (bits @c 5 through @c 0) directly affect the maximum number
* of characters parsed for this particular parameter and also imply the size
* of the data storage pointed to by @c data_ptr. Their exact meaning depends
* on @link ParamValue::type type@endlink. For instance, if @c type is set to
* #DTYPE_UINT and the size bits are set to 0x08, the assigned parser should
* parse the stream for an integer value of up to 255, storing it at the
* memory location pointed to by @c data_ptr. Note, that in case of
* #DTYPE_UINT, not all size values would make sense (eg, 10). Ultimately, it
* depends on the parser.
*
* As another example, setting @c type to #DTYPE_STRING and the size bits to
* 0x10, direct the assigned parser to copy *up to 15* bytes from the input
* stream to the buffer pointed to by @c data_ptr followed by one final
* null-byte. Whether 15 or less bytes are actually copied depends on the
* semantics supported by the parser and what, for example, constitutes an
* appropriate delimiter.
*
* In order to ensure smooth operation between successive invocations and to
* avoid redundant initializations, the parsers should make certain to never
* alter the contents of the size bits.
*/
uint8_t status_len;
} ParamValue;

/**
* @brief Wrapper of parameter-tokens and their #ParamValue.
*/
typedef struct {
    /** @brief Array of parameter tokens to match against the stream. */
    uint8_t** tokens;

    /** @brief Array of #ParamValue corresponding to @c tokens. */
    ParamValue* values;

    /**
     * @brief The size of @c tokens and @c values.
     *
     * Normally, @c tokens and @c values are of the same size (since they are

```

```
* related one-on-one).
*/
uint8_t len;
} ParamInfo;

/**
 * @brief Extract token status out of @link ParamValue::status_len@endlink.
 *
 * Once applied to @c status_len, it preserves just the part that corresponds to
 * the status. The result may then be compared against #PARAM_NOT_SET,
 * #PARAM_VALID, #PARAM_INVALID and/or #PARAM_TOO_LONG. It may also be used to
 * reset the status (to indicate it has not been set yet). This is done
 * automatically by json_parse() for the tokens supplied for parsing.
 */
#define PARAM_STATUS_MASK (0xCO)

/**
 * @brief Indicates that this token has not occurred on the stream.
 *
 * This only holds true if the corresponding bits of @c status_len have initially
 * been set to this particular value. This is done automatically by json_parse()
 * for the tokens supplied for parsing.
 *
 * Note that this value refers only to the two most significant bits of @c
 * status_len. For details, see #ParamValue.
 *
 * Also, see #PARAM_STATUS_MASK, #PARAM_VALID, #PARAM_INVALID, #PARAM_TOO_LONG.
 */
#define PARAM_NOT_SET (0x00)

/**
 * @brief Indicates that an invalid character occurred during parsing for a
 * value.
 *
 * Note that this value refers only to the two most significant bits of @c
 * status_len. For details, see #ParamValue.
 *
 * Also, see #PARAM_STATUS_MASK, #PARAM_NOT_SET, #PARAM_VALID, #PARAM_TOO_LONG.
 */
#define PARAM_INVALID (0x40)

/**
 * @brief Indicates that too many characters have been read during parsing for a
 * value.
 *
 * Note that this value refers only to the two most significant bits of @c
 * status_len. For details, see #ParamValue.
 *
```

```

* Also, see #PARAM_STATUS_MASK, #PARAM_NOT_SET, #PARAM_VALID, #PARAM_INVALID.
*/
#define PARAM_TOO_LONG (0x80)

/***
* @brief Indicates that a valid value has been parsed from the stream.
*
* The value is stored into @link ParamValue::data_ptr@endlink.
*
* Note that this value refers only to the two most significant bits of @c
* status_len. For details, see #ParamValue.
*
* Also, see #PARAM_STATUS_MASK, #PARAM_NOT_SET, #PARAM_INVALID, #PARAM_TOO_LONG.
*/
#define PARAM_VALID (0xC0)

/***
* @brief Condition that checks whether a parameter has been set.
*
* @param[in] var Name of a (local) #ParamValue array.
* @param[in] x Index of @p var to check.
*/
#define PARAM_IS_SET(var, x) \
((var[x].status_len & PARAM_STATUS_MASK) == PARAM_VALID)

/***
* @brief Facilitates initialisation of an 8-bit #DTYPE_UINT #ParamValue.
*
* @param[in] x The variable (and *not* its address) which the parsed value will
* be parsed into.
*/
#define PARAM_UINT8(x) \
{.type = DTYPE_UINT, .data_ptr = &x, .status_len = 8}

/***
* @brief Facilitates initialisation of a #DTYPE_STRING #ParamValue.
*
* @param[in] x The array (address of its first byte) which the parsed string
* will be parsed into.
* @param[in] len Maximum acceptable length of parsed string (inclusive of
* null-byte).
*/
#define PARAM_STRING(x, len) \
{.type = DTYPE_STRING, .data_ptr = x, .status_len = len}

/***
* @brief Directive to flush the serialised parameters.
*/

```

```

#define SERIAL_FLUSH (1 << 7)

/**
 * @brief Directive to create a new parameter wrapper.
 */
#define SERIAL_ATOMIC_S (1 << 6)

/**
 * @brief Directive to end a parameter wrapper.
 */
#define SERIAL_ATOMIC_E (1 << 5)

/**
 * @brief Directive to create a parameter envelope.
 */
#define SERIAL_ENVELOPE_S (1 << 4)

/**
 * @brief Directive to end a parameter envelope.
 */
#define SERIAL_ENVELOPE_E (1 << 3)

/**
 * @brief Directive to inform that previous parameters have been serialised.
 */
#define SERIAL_PRECEDED (1 << 2)

/**
 * @brief Directive to serialise and flush parameters as one complete group.
 */
#define SERIAL_DEFAULT \
(SERIAL_ATOMIC_S \
| SERIAL_ATOMIC_E \
| SERIAL_FLUSH)

#endif /* PARAM_H_INCL */

```

## Περιγραφή πόρων HTTP

### resource.h

```

/**
 * @file
 * @addtogroup resource Server Resource
 * @ingroup http_server
 */

#ifndef RESOURCE_H_INCL
#define RESOURCE_H_INCL

```

```
struct HTTPRequest;

#include "param.h"
#include "defs.h"

#include <inttypes.h>

/** 
 * @brief Index of / in #rsrc_handlers.
 *
 * Relative path of a virtual file.
 */
#define RSRC_ROOT 1

/** 
 * @brief Index of /client.js in #rsrc_handlers.
 *
 * Relative path of a virtual file.
 */
#define RSRC_CLIENT_JS 2

/** 
 * @brief Index of /index in #rsrc_handlers.
 *
 * Relative path of a virtual file.
 */
#define RSRC_INDEX 5

/** 
 * @brief Index of /logo.png in #rsrc_handlers.
 *
 * Relative path of a virtual file.
 */
#define RSRC_LOGO_PNG 6

/** 
 * @brief Index of /measurement in #rsrc_handlers.
 *
 * Provides access to query string parameters.
 */
#define RSRC_MEASUREMENT 7

/** 
 * @brief Index of /style.css in #rsrc_handlers.
 *
 * Relative path of a virtual file.
 */

```

```
#define RSRC_STYLE_CSS 8

/**
 * @brief Specification of methods that trigger a particular callback function.
 *
 * The callback function receives a pointer to the #HTTPRequest representation of
 * the current request and may work with parsers to identify data on the input
 * stream and produce an appropriate response. It is safe to assume the callback
 * is only executed if the requested method (HTTPRequest::method) is applicable
 * to this particular instance (ie, it has a corresponding bit set in
 * @link ResourceHandler::methods methods@endlink).
 *
 * Note that @link ResourceHandler::methods methods@endlink is the bit-wise OR
 * result of #MethodFlag values while #HTTPRequest::method is an *index* number
 * of a #server_consts string literal.
 */
typedef struct ResourceHandler {
    /**
     * @brief Bit-wise-OR'ed #MethodFlag values representing the acceptable
     * methods for this resource. */
    uint8_t methods;

    /** @brief Handler callback for this particular path. */
    void (*call)(struct HTTPRequest*);
} ResourceHandler;

/**
 * @brief Container of query parameters and their values.
 *
 * This is used by the http_parser module <http_parser.h> to identify the
 * acceptable parameters for a particular URI and method combination. It is also
 * set to contain their value as they are found on the stream. It is imperative
 * most members be set up before use. Generally, the necessary initialisation is
 * performed by rsrc_inform().
 */
typedef struct QueryString {
    /**
     * @brief Array of acceptable query parameter names.
     *
     * The items of this array relate to the items of .values on a one-on-one
     * basis; the string found in the corresponding index of .values is the value
     * of the parameter token found at the same index in this array.
     *
     * Typically, the acceptable parameters (tokens) are specified to the
     * http_parser module *after* the targeted resource of the request has been
     * found to match one of ServerSettings#rsrc_tokens. rsrc_inform() may be
     * used to update the contents of this member at that time.
     */
    uint8_t* tokens[QUERY_PARAM_LEN];
```

```

/**
 * @brief Values for the parameters found in .tokens.
 *
 * The items of this array relate to the items of .tokens on a one-on-one
 * basis; the string at any index of this array is the value that has been
 * specified in the query string for the parameter token that resides at the
 * same index in array .tokens. @c NULL denotes that the parameter was not
 * found in the query string. */
uint8_t* values[QUERY_PARAM_LEN];

/**
 * @brief The permissible number of parameters for a particular resource.
 *
 * Typically, the acceptable parameters are specified *after* the resource of
 * the request has been identified to match one of
 * ServerSettings#rsrc_tokens. rsrc_inform() may be used to update the
 * contents of this member at that time.
 * This represents the maximum permissible number of parameters for a
 * particular resource which may or may not be equal to the size of arrays
 * .tokens and .values.
 *
 * A value of @c 0 denotes that no parameters are expected in the query
 * string.
 */
uint8_t count;

/**
 * @brief Stores query parameter tokens and values.
 *
 * String pointers in .tokens and .values are set to addresses found within
 * this buffer or, alternatively, @c NULL (in case of .values).
 */
uint8_t buf[QUERY_BUF_LEN];

/** @brief Offset in .buf to write to next. */
uint16_t buf_i;

/** @brief The size of .buf. */
uint16_t buf_len;
} QueryString;

/**
 * @brief Initialise the Resource module.
 *
 * This basically registers the resource tokens and handler function to the HTTP
 * server so that they may be invoked every time a resource token is matched.
 */

```

```
* It suffices to call this only once; even if a particular handler is replaced
* by another, there is no need for an update.
*/
void rsrc_init();

void rsrc_set_parser(int8_t (*parser)(uint8_t**, ParamValue*, uint8_t len));

void rsrc_set_serial(void (*serialiser)(uint8_t**, ParamValue*, uint8_t len, uint8_t ctr));


```

```

* uri is received.
*/
void rsrc_set_handler(uint8_t uri,
                      uint8_t methods,
                      void (*handler)(struct HTTPRequest*));

/**
* @brief Update @p req with URI-method specific options.
*
* Apart from server-wide options that are readily available, options closely
* linked to a particular resource may only be provided once such a resource has
* been identified. This is done to conserve main memory that would otherwise be
* consumed for all resource-specific tokens, regardless of them being used or
* not. One such example is the query string parameters. Only the appropriate
* tokens are loaded into main memory for a particular URI-method pair.
*
* Currently, this function is a wrapper around rsrc_get_qparam().
*
* @param[in,out] req An #HTTPRequest variable that has its .uri and .method
* members set. The appropriate members of @p req will be initialised.
*/
void rsrc_inform(struct HTTPRequest* req);

/**
* @brief Update the #QueryString of @p req according to .uri and .method.
*
* It does the following:
* — The appropriate query parameter tokens are loaded into main memory, and in
* particular, in @link QueryString#buf query.buf@endlink.
* — @link QueryString#count query.count@endlink is updated to reflect the
* permissible amount of parameters, the tokens of which can be found in
* @link QueryString#tokens query.tokens@endlink.
* — Indices in @link QueryString#values query.values@endlink equal to the amount
* of permissible parameters are set to @c NULL to indicate they have not yet
* been given a value.
* — @link QueryString#buf_i query.buf_i@endlink is set to an offset within
* @link QueryString#buf query.buf@endlink, which the first parameter value may
* be written at.
* — @link QueryString#buf_len query.buf_len@endlink is set to #QUERY_BUF_LEN.
*
* @param[in,out] req Accepts an #HTTPRequest variable that has its
* @link HTTPRequest#uri uri@endlink and
* @link HTTPRequest#method method@endlink members set. If no parameters are
* applicable to them, @link QueryString#count query.count@endlink will be set
* to @c 0 whereas @link QueryString#buf_i query.buf_i@endlink will be equal to
* @link QueryString#buf_len query.buf_len@endlink (meaning the buffer is full
* and should not be written to).
*/

```

```
static inline void rsrc_get_qparam(struct HTTPRequest* req);

#endif /* RESOURCE_H_INCL */

resource.c

#include "resource.h"
/* See #include "resource_handlers.inc" further below. */
#include "param.h"
#include "http_server.h"

#include <stdio.h>
#include <inttypes.h>

/** 
 * @ingroup resource
 * @brief The number of token–handler pairs in #rsrc_handlers.
 */
#define RSRC_LEN 9

/** 
 * @ingroup resource
 * @brief Function pointer to a parser conforming to the param.h module.
 *
 * This is set using rsrc_set_parser() and it should be set before any resource
 * handlers are invoked.
 */
static int8_t (*parser)(uint8_t**, ParamValue*, uint8_t);

/** 
 * @ingroup resource
 * @brief Serialising function pointer conforming to the param.h module.
 *
 * It is set using rsrc_set_serial() and it should be set before any resource
 * handlers are invoked.
 */
static void (*serialiser)(uint8_t**, ParamValue*, uint8_t, uint8_t);

/*
 * This module is divided into two parts; this file, containing common base
 * functions, and resource_handlers.inc containing the definition of each
 * resource handler. The latter is included after #parser and #serialiser have
 * been declared because they are needed in the included source code.
 * resource_handlers.inc should not be compiled separately.
 */
#include "resource_handlers.inc"

/**
```

```

* @ingroup resource
* @brief Absolute path tokens of resources exposed by the HTTP server.
*/
static uint8_t* rsrc_tokens[RSRC_LEN] = {
    "*",
    "/",
    "/client.js",
    "/configuration",
    "/coordinates",
    "/index",
    "/logo.png",
    "/measurement",
    "/style.css"
};

/** 
* @ingroup resource
* @brief Available methods and corresponding handlers for each resource.
*
* This array and #rsrc_tokens are correlated through srvr_set_resources().
*/
static ResourceHandler rsrc_handlers[RSRC_LEN]; /* Definition is in the end. */

void rsrc_init() {
    srvr_set_resources(rsrc_tokens, rsrc_handlers, RSRC_LEN);
}

void rsrc_set_parser(int8_t
                     (*new_parser)(uint8_t**, ParamValue*, uint8_t len)) {
    parser = new_parser;
}

void rsrc_set_serial(void (*new_serialiser)(uint8_t**,
                                             ParamValue*,
                                             uint8_t len,
                                             uint8_t ctr)) {
    serialiser = new_serialiser;
}

void rsrc_set_handler(uint8_t uri,
                      uint8_t methods,
                      void (*handler)(HTTPRequest*)) {

    if(uri < RSRC_LEN) {
        rsrc_handlers[uri].methods = methods;
        rsrc_handlers[uri].call = handler;
    }
}

```

```
void rsrc_inform(struct HTTPRequest* req) {
    rsrc_get_qparam(req);
}

static inline void rsrc_get_qparam(struct HTTPRequest* req) {
    uint16_t offset; /* QueryString.buf_i. */
    uint8_t i; /* Number of permissible parameters. */

    if(req->uri == RSRC_MEASUREMENT && req->method == METHOD_GET) {
        i = 4;
        offset = pgm_read_str_array(req->query.tokens,
                                    req->query.buf,
                                    /* These string addresses are defined in resource_handlers.inc. */
                                    prm_date_since,
                                    prm_date_until,
                                    prm_page_index,
                                    prm_page_size,
                                    NULL);

    } else {
        i = 0;
        offset = QUERY_BUF_LEN;
    }

    req->query.buf_len = QUERY_BUF_LEN;
    req->query.buf_i = offset;

    req->query.count = i;

    /* Reset .values to a known value to indicate they have not yet been set.
     * This is used in place of memset(). */
    while(i) {
        --i;
        req->query.values[i] = NULL;
    }
}

static ResourceHandler rsrc_handlers[RSRC_LEN] = {
    /* server */
    {.methods = HTTP_OPTIONS, .call = &rsrc_handle_server},
    /* root */
    {.methods = HTTP_GET, .call = &rsrc_handle_file},
    /* client.js */
    {.methods = HTTP_GET, .call = &rsrc_handle_file},
    /* configuration */
    {.methods = HTTP_GET
     | HTTP_PUT, .call = &rsrc_handle_configuration},
}
```

```

/* coordinates */
{.methods = HTTP_GET
 | HTTP_PUT, .call = &rsrc_handle_coordinates},
/* index */
{.methods = HTTP_GET, .call = &rsrc_handle_file},
/* logo.png */
{.methods = HTTP_GET, .call = &rsrc_handle_file},
/* /measurement */
{.methods = HTTP_GET
 | HTTP_POST, .call = &rsrc_handle_measurement},
/* style.css */
{.methods = HTTP_GET, .call = &rsrc_handle_file}
};

```

## Pouτίνες περάτωσης υλοποίησης

### **resource\_handlers.inc**

```

/*
* This module (resource) is divided into two parts; resource.c, containing
* common base functions, and this file, containing the definition of each
* resource handler. It is assumed that #parser and #serialiser have already been
* defined as part of resource.c and are currently available.
*/

#include "http_server.h"
#include "defs.h"
#include "param.h"
#include "util.h"
#include "motor.h"
#include "rtc.h"
#include "log.h"
#include "task.h"
#include "w5100.h"

#include <avr/pgmspace.h>
#include <inttypes.h>

/**
* @ingroup resource
* @brief Index of parameter "date" (in resource /configuration).
*/
#define PRM_SRVR_DATE 0

/**
* @ingroup resource
* @brief Index of parameter "day" (in resource /configuration).
*/
#define PRM_SRVR_DAY 1

```

```
/**  
 * @ingroup resource  
 * @brief Index of parameter "gateway" (in resource /configuration).  
 */  
#define PRM_SRVR_GATEWAY 2  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "iaddr" (in resource /configuration).  
 */  
#define PRM_SRVR_IADDR 3  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "interval" (in resource /configuration).  
 */  
#define PRM_TASK_INTERVAL 4  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "samples" (in resource /configuration).  
 */  
#define PRM_TASK_SAMPLES 5  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "subnet" (in resource /configuration).  
 */  
#define PRM_SRVR_SUBNET 6  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "x" (in resource /configuration).  
 */  
#define PRM_SRVR_X 7  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "y" (in resource /configuration).  
 */  
#define PRM_SRVR_Y 8  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "z" (in resource /configuration).  
 */  
#define PRM_SRVR_Z 9
```

```
/**  
 * @ingroup resource  
 * @brief Size of a date string (ISO8601 format) (inclusive of null-byte).  
 */  
#define PRM_DATE_LEN 25  
  
/**  
 * @ingroup resource  
 * @brief Size of any IPv4 address string (inclusive of null-byte).  
 */  
#define PRM_INET_LEN 16  
  
/**  
 * @ingroup resource  
 * @brief Size of any temperature reading (inclusive of null-byte).  
 */  
#define PRM_TEMP_LEN 6  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "date-since" (in resource /measurement).  
 */  
#define PRM_MSR_DATE_SINCE 0  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "date-until" (in resource /measurement).  
 */  
#define PRM_MSR_DATE_UNTIL 1  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "page-index" (in resource /measurement).  
 */  
#define PRM_MSR_PAGE_INDEX 2  
  
/**  
 * @ingroup resource  
 * @brief Index of parameter "page-size" (in resource /measurement).  
 */  
#define PRM_MSR_PAGE_SIZE 3  
  
#ifndef BV  
/**  
 * @brief Set a single bit.  
 *  
 * @param[in] x The bit to set.
```

```
/*
#define BV(x) (1 << x)
#endif

/*
 * @brief Token: x
 *
 * This is used by rsrc_handle_coordinates() to read and return device
 * coordinates.
*/
static uint8_t prm_x[] PROGMEM = "x";

/* Doxygen does not handle attributes (like PROGMEM) very well. */
/*
 * @brief Token: y
 *
 * This is used by rsrc_handle_coordinates() to read and return device
 * coordinates.
*/
static uint8_t prm_y[] PROGMEM = "y";

/*
 * @brief Token: z
 *
 * This is used by rsrc_handle_coordinates() to read and return device
 * coordinates.
*/
static uint8_t prm_z[] PROGMEM = "z";

/*
 * @brief Token: date
 *
 * This is used by rsrc_handle_configuration() to parse a new or return the
 * current server date and time.
*/
static uint8_t prm_date[] PROGMEM = "date";

/*
 * @brief Token: day
 *
 * This is used by rsrc_handle_configuration() to parse a new or return the
 * current server day (eg, @c 1 for Sunday).
*/
static uint8_t prm_day[] PROGMEM = "day";

/*
 * @brief Token: gateway
 *
```

```
* This is used by rsrc_handle_configuration() to parse a new or return the
* current server default gateway.
*/
static uint8_t prm_gateway[] PROGMEM = "gateway";

/*
* @brief Token: ip
*
* This is used by rsrc_handle_configuration() to parse a new or return the
* current server IP address.
*/
static uint8_t prm_iaddr[] PROGMEM = "iaddr";

/*
* @brief Token: interval
*
* This is used by rsrc_handle_configuration() to parse a new or return the
* current sampling interval.
*/
static uint8_t prm_interval[] PROGMEM = "interval";

/*
* @brief Token: samples
*
* This is used by rsrc_handle_configuration() to parse a new or return the
* current amount of samples after each interval.
*/
static uint8_t prm_samples[] PROGMEM = "samples";

/*
* @brief Token: subnet
*
* This is used by rsrc_handle_configuration() to parse a new or return the
* current server subnet mask.
*/
static uint8_t prm_subnet[] PROGMEM = "subnet";

/*
* @brief Token: page-index
*
* It specifies the page of search results returned by
* rsrc_handle_measurement().
*/
static uint8_t prm_page_index[] PROGMEM = "page-index";

/*
* @brief Token: page-size
*
```

```
* It specifies the number of search results returned by
* rsrc_handle_measurement().
*/
static uint8_t prm_page_size[] PROGMEM = "page-size";

/*
* @brief Token: date-since
*
* It specifies the starting date of search results returned by
* rsrc_handle_measurement().
*/
static uint8_t prm_date_since[] PROGMEM = "date-since";

/*
* @brief Token: date-until
*
* It specifies the end date of search results returned by
* rsrc_handle_measurement().
*/
static uint8_t prm_date_until[] PROGMEM = "date-until";

/*
* @brief Token: log
*
* It contains the records returned by rsrc_handle_measurement().
*/
static uint8_t prm_log[] PROGMEM = "log";

/*
* @brief Token: total
*
* It specifies the total amount of available records to be returned by
* rsrc_handle_measurement().
*/
static uint8_t prm_total[] PROGMEM = "total";

/*
* @brief Token: ph
*
* The pH value of a record, as returned by rsrc_handle_measurement().
*/
static uint8_t prm_ph[] PROGMEM = "ph";

/*
* @brief Token: rh
*
* The Relative Humidity value of a record, as returned by
* rsrc_handle_measurement().
*/
```

```
/*
static uint8_t prm_rh[] PROGMEM = "rh";

/*
 * @brief Token: t
 *
 * The Temperature value of a record, as returned by rsrc_handle_measurement().
 */
static uint8_t prm_t[] PROGMEM = "t";

void rsrc_handle_server(HTTPRequest* req) {

    /**
     * @brief Manage device files.
     *
     * Currently, only method GET is supported to retrieve the
     * Method GET:
     * Returns the file specified in the URI. The available files are:
     * — index
     * — style.css
     * — client.js
     * — logo.png
     * Typically, only @c index needs to be loaded explicitly; the others are
     * requested automatically by the browser. Note that both / and /index are
     * equivalent.
     */
    void rsrc_handle_file(HTTPRequest* req) {
        uint16_t page = 0;
        uint16_t size = 0;
        uint8_t is_gzip = 1;

        srvr_prep(TXF_STATUS_200, TXF_ln,
                  TXF_STANDARD_HEADERS_ln);

        switch(req->uri) {
            case RSRC_CLIENT_JS:
                srvr_prep(TXF_CONTENT_TYPE_JS_ln);
                page = FILE_PAGE_CLIENT_JS;
                size = FILE_SIZE_CLIENT_JS;
                break;

            case RSRC_ROOT:
            case RSRC_INDEX:
                srvr_prep(TXF_CONTENT_TYPE, TXF_HS,
                          TXFx_FROMRAM, MIME_MIN + MIME_TEXT_HTML, TXF_ln);
                page = FILE_PAGE_INDEX;
                size = FILE_SIZE_INDEX;
        }
    }
}
```

```
        break;

    case RSRC_STYLE_CSS:
        srvr_prep(TXF_CONTENT_TYPE_CSS_ln);
        page = FILE_PAGE_STYLE_CSS;
        size = FILE_SIZE_STYLE_CSS;
        break;

    case RSRC_LOGO_PNG:
        is_gzip = 0;
        srvr_prep(TXF_CONTENT_TYPE, TXF_HS,
                   TXFx_FW_STRING, "image/png", TXF_ln);
        page = FILE_PAGE_LOGO_PNG;
        size = FILE_SIZE_LOGO_PNG;
        break;
    }

    if(is_gzip) {
        srvr_prep(TXF_GZIP_ln);
    }
    srvr_prep(TXF_CACHE_PUBLIC_ln,
              TXF_CONTENT_LENGTH, TXF_HS, TXFx_FW_UINT, size, TXF_lnl);
}

fls_to_wiz(HTTP_SOCKET, page, size);
net_send(HTTP_SOCKET, NULL, 0, 1);
}

/**
 * @ingroup resource
 * @brief Manage device configuration.
 *
 * Method GET:
 * Returns the current configuration of the device. The response status code is
 * always 200. Format: @verbatim {
 * "date" : string, // Format ISO8601: YYYY-MM-DDTHH:mm:ss[.sss][Z]
 * "day" : number, // Day-of-the-week; 1 denotes Sunday
 * "gateway" : string, // Server default gateway (dot-notation)
 * "iaddr" : string, // Server IP address
 * "interval" : number, // Time between successive samplings; up to 240
 * "samples" : number // Amount of samplings to perform each time
 * "subnet" : string, // Server subnet mask (xxx.xxx.xxx.xxx)
 * "x" : number, // Custom maximum X dimension
 * "y" : number, // Custom maximum Y dimension
 * "z" : number // Custom maximum Z dimension
} @endverbatim
 * Additional notes:
 * - Fraction of second (in @c date) is always returned as @c 000 and never
 * parsed. Time-zone is UTC (@c Z).
```



```
PARAM_UINT8(task.interval),
PARAM_UINT8(task.samples),
PARAM_STRING(s_subnet, PRM_INET_LEN),
PARAM_UINT8(max.x),
PARAM_UINT8(max.y),
PARAM_UINT8(max.z);}

/* Load tokens into main memory. */
pgm_read_str_array(tokens, token_buf, prm_date,
                    prm_day,
                    prm_gateway,
                    prm_iaddr,
                    prm_interval,
                    prm_samples,
                    prm_subnet,
                    prm_x,
                    prm_y,
                    prm_z, NULL);

/* ---- INITIALISATION end --- */

uint8_t status; /* Status of response. */
uint16_t size = 167; /* Response size without Inet address values. */
uint8_t iaddr[4]; /* Numerical IP address. */
uint8_t subnet[4]; /* Numerical subnet mask. */
uint8_t gateway[4]; /* Numerical default gateway address. */
BCDDate dt; /* Date and time. */
uint8_t set_params = 0; /* Flags of parameters that have been set (mostly
* for PRM_SRVR_*. */

/* Load current maximum position. */
sys_get(SYS_MTR_MAX, &max);

/* Load current task settings. */
sys_get(SYS_TASK, &task);

/* Load current server configuration. */
sys_get(SYS_GATEWAY, gateway);
sys_get(SYS_IADDR, iaddr);
sys_get(SYS_SUBNET, subnet);
status = TXF_STATUS_200;

if(req->method == METHOD_PUT) {
    if(!(*parser)(tokens, params, 10)) {

        /* Date and day must both be set, if any one of them is present. */
        if(PARAM_IS_SET(params, PRM_SRVR_DATE)
        && PARAM_IS_SET(params, PRM_SRVR_DAY)) {
```

```

        if(str_to_date(&dt, s_date) || day > 7) {
            status = TXF_STATUS_400;

            /* Set date to RTC and remove records more recent than @c dt. */
            } else {
                log_purge(&dt);
                set_date(&dt, day);
            }
        }

        /* If any one of PRM_SRVR_GATEWAY, IADDR or SUBNET has been set,
         * it is parsed into numbers but only applied *after* the response
         * has been sent out with the current configuration. */
        if(PARAM_IS_SET(params, PRM_SRVR_GATEWAY)) {
            if(str_to_inet(gateway,
                           (uint8_t*)params[PRM_SRVR_GATEWAY].data_ptr)) {
                status = TXF_STATUS_400;
            } else {
                set_params |= BV(PRM_SRVR_GATEWAY);
            }
        }

        if(PARAM_IS_SET(params, PRM_SRVR_IADDR)) {
            if(str_to_inet(iaddr,
                           (uint8_t*)params[PRM_SRVR_IADDR].data_ptr)) {
                status = TXF_STATUS_400;
            } else {
                set_params |= BV(PRM_SRVR_IADDR);
            }
        }

        if(PARAM_IS_SET(params, PRM_SRVR_SUBNET)) {
            if(str_to_inet(subnet,
                           (uint8_t*)params[PRM_SRVR_SUBNET].data_ptr)) {
                status = TXF_STATUS_400;
            } else {
                set_params |= BV(PRM_SRVR_SUBNET);
            }
        }

        /* New maximum coordinates may be specified at any time; the motors
         * will automatically reset. */
        if(PARAM_IS_SET(params, PRM_SRVR_X)
        || PARAM_IS_SET(params, PRM_SRVR_Y)
        || PARAM_IS_SET(params, PRM_SRVR_Z)) {

            /* If the physical limits of the device are exceeded, return the

```

```

        * absolute maximum values. */
    if(motor_set_max(&max)) {
        status = TXF_STATUS_400;
    } else {
        sys_set(SYS_MTR_MAX, &max);
    }
}

if(PARAM_IS_SET(params, PRM_TASK_INTERVAL)
|| PARAM_IS_SET(params, PRM_TASK_SAMPLES)) {
    if(task.interval > TASK_INTERVAL_MAX) {
        status = TXF_STATUS_400;
    } else {
        sys_set(SYS_TASK, &task);
    }
}

} else {
    status = TXF_STATUS_400;
}
}

switch(status) {
case TXF_STATUS_200:

    size += inet_to_str(s_gateway, gateway);
    size += inet_to_str(s_iaddr, iaddr);
    size += inet_to_str(s_subnet, subnet);
    get_date(&dt, &day);
    date_to_str(s_date, &dt);

    srvr_prep(TXF_STATUS_200, TXF_ln,
              TXF_STANDARD_HEADERS_ln,
              TXF_CONTENT_TYPE_JSON_ln,
              TXF_CACHE_NO_CACHE_ln,
              TXF_CONTENT_LENGTH, TXF_HS, TXFx_FW_UINT, size,
              TXF_lnl);

    (*serialiser)(tokens, params, 10, SERIAL_DEFAULT);

    /* Apply changes to the address after the response has been sent. */
    if(set_params) {
        if(set_params & BV(PRM_SRVR_GATEWAY)) {
            sys_set(SYS_GATEWAY, gateway);
        }
        if(set_params & BV(PRM_SRVR_IADDR)) {
            sys_set(SYS_IADDR, iaddr);
        }
    }
}

```

```

        if(set_params & BV(PRM_SRVR_SUBNET)) {
            sys_set(SYS_SUBNET, subnet);
        }
    }

break;
case TXF_STATUS_400:

    /* On any error, return maximum allowable values, by default. */
    max.x = GRID_X_LEN;
    max.y = GRID_Y_LEN;
    max.z = GRID_Z_LEN;
    task.interval
        = TASK_INTERVAL_MAX;

    srvr_send(TXF_STATUS_400, TXF_ln,
              TXF_STANDARD_HEADERS_ln,
              TXF_CONTENT_TYPE_JSON_ln,
              TXF_CACHE_NO_CACHE_ln,
              TXF_CONTENT_LENGTH, TXF_HS, TXFx_FW_UINT, 57,
              TXF_lnl);

    (*serialiser)(&tokens[PRM_TASK_INTERVAL],
                  &params[PRM_TASK_INTERVAL], 1, SERIAL_ATOMIC_S);
    (*serialiser)(&tokens[PRM_SRVR_X],
                  &params[PRM_SRVR_X], 3, SERIAL_PRECEDED
                  | SERIAL_ATOMIC_E
                  | SERIAL_FLUSH);

    break;
}

/** 
 * @ingroup resource
 * @brief Manage device positioning.
 *
 * Method GET:
 * Returns the current coordinates of the device head.
 * — 200 OK; the body contains the current coordinates of the head. Format:
 * @verbatim {
        'x' : number,
        'y' : number,
        'z' : number
} @endverbatim
 * — 503 Service Unavailable; the device is currently positioning itself. The
 * request should be reattempted later, as specified in the 'Retry-After'
 * header.
 *

```

```
* Method PUT:  
* Moves the device head to the specified coordinates.  
* The message body should include any of the above specified keys, the order of  
* which is irrelevant, and except for @c z. @c z may only be implicitly set when  
* a measurement is requested (see rsrc_handle_measurement()), and is  
* automatically retracted once the measurement is complete. For @c x and @c y,  
* only the last instance of each key is preserved. Returns:  
* - 200 OK; the device is already at the specified position -- no action will  
* be taken. The body contains the current coordinates. The format is the  
* same as above.  
* - 202 Accepted; the specified coordinates were valid and the device's  
* repositioning has been initiated. Header 'Retry-After' designates the  
* estimated time until completion.  
* - 400 Bad request; the specified coordinates lay outside the allowable  
* device-space or an invalid parameter and/or value has been specified.  
* The body contains the maximum acceptable values for each axis. The  
* format is the same as above.  
* - 503 Service Unavailable; same as with GET.  
*/  
void rsrc_handle_coordinates(HTTPRequest* req) {  
    uint16_t eta; /* Time until pending tasks complete. */  
    Position pos;  
  
    /* Position reading and updating may only be performed if the motors are  
     * not being operated. In either case, the current position is read and  
     * either returned (in case of GET), or updated with a new value (in case of  
     * PUT). Return 503 Service Unavailable, otherwise. */  
    if(motor_get(&pos)) {  
        eta = task_get_estimate();  
  
        srvr_send(TXF_STATUS_503, TXF_ln,  
                  TXF_STANDARD_HEADERS_ln,  
                  TXF_CACHE_NO_CACHE_ln,  
                  TXF_CONTENT_LENGTH_ZERO_ln,  
                  TXF_RETRY_AFTER, TXF_HS, TXFx_FW_UINT, eta,  
                  TXF_lnl);  
        return;  
    }  
  
    uint8_t status; /* Status of response. */  
    uint8_t token_buf[6]; /* Key tokens. */  
    uint8_t* tokens[3]; /* Pointers to each token in @c token_buf. */  
    Position npos = pos; /* New position. */  
  
    ParamValue params[] = {PARAM_UINT8(npos.x),  
                          PARAM_UINT8(npos.y),  
                          PARAM_UINT8(npos.z)};
```

```

pgm_read_str_array(tokens, token_buf, prm_x, prm_y, prm_z, NULL);

if(req->method == METHOD_PUT) {

    /* If an acceptable set of parameters have been parsed, attempt to use
     * @c pos to update motor position. It could still fail if, for example,
     * an out-of-range value has been specified. */
    int8_t retval;

    /* Accept a value only for the x and y coordinates. */
    if(!retval = json_parse(tokens, params, 2))) {

        /* Already there. */
        if(pos.x == npos.x && pos.y == npos.y) {
            status = TXF_STATUS_200;

            /* Invalid coordinate (out of bounds). */
        } else if(motor_set(npos)) {
            status = TXF_STATUS_400;

            /* Position is valid and will be processed. Respond with a 202
             * Accepted and a ‘Retry-Later’ header to indicate the estimated
             * completion time. */
        } else {
            status = TXF_STATUS_202;
            eta = task_get_estimate(); /* Update estimate. */
        }

        /* Wrong argument. */
    } else {
        status = TXF_STATUS_400;
    }

} else if(req->method == METHOD_GET) {
    status = TXF_STATUS_200;
}

switch(status) {
    case TXF_STATUS_200:
        srvr_prep(TXF_STATUS_200, TXF_ln,
                  TXF_STANDARD_HEADERS_ln,
                  /* This should be using req->accept, after it is set
                   * to specific type (ie, not app/* but app/json). */
                  TXF_CONTENT_TYPE_JSON_ln,
                  TXF_CACHE_NO_CACHE_ln,
                  TXF_CONTENT_LENGTH, TXF_HS, TXFx_FW_UINT, 38,
                  TXF_lnl);
        (*serialiser)(tokens, params, 3, SERIAL_DEFAULT);
}

```

```
break;
case TXF_STATUS_202:
    srvr_send(TXF_STATUS_202, TXF_ln,
              TXF_STANDARD_HEADERS_ln,
              TXF_CACHE_NO_CACHE_ln,
              TXF_CONTENT_LENGTH_ZERO_ln,
              TXF_RETRY_AFTER, TXF_HS, TXFx_FW_UINT, eta,
              TXF_lnl);
break;
case TXF_STATUS_400:
    srvr_send(TXF_STATUS_400, TXF_ln,
              TXF_STANDARD_HEADERS_ln,
              TXF_CONTENT_TYPE_JSON_ln,
              TXF_CACHE_NO_CACHE_ln,
              TXF_CONTENT_LENGTH, TXF_HS, TXFx_FW_UINT, 38,
              TXF_lnl);
/* Return maximum values. */
sys_get(SYS_MTR_MAX, &npos);

(*serialiser)(tokens, params, 3, SERIAL_DEFAULT);
break;
case TXF_STATUS_503:
    /* This is implemented as a guard statement in the beginning of the
     * the function. */
    break;
}
}

/**
 * @brief Serialise statistical information of records to be returned.
 *
 * The result is similar to this: @verbatim {
 * "page-index": number,
 * "page-size" : number,
 * "total" : number
 @endverbatim
 *
 * No flushing is performed.
 *
 * @param[in] page_index The page of results.
 * @param[in] page_size The (maximum) number of records within this
 * page/response.
 * @param[in] total The amount of available records (regardless of pagination).
 */
static void rsrc_measurement_serial_info(uint8_t page_index,
```

```

        uint8_t page_size,
        uint8_t total) {

    uint8_t token_buf[31]; /* Key tokens. */
    uint8_t* tokens[4]; /* Pointers to each token in @c token_buf. */

    ParamValue params[] = {PARAM_UINT8(page_index),
                           PARAM_UINT8(page_size),
                           PARAM_UINT8(total)};

    /* Load tokens into main memory. */
    pgm_read_str_array(tokens, token_buf, prm_page_index,
                        prm_page_size,
                        prm_total,
                        prm_log, NULL);

    (*serialiser)(tokens, params, 3, SERIAL_ATOMIC_S);

    /* An envelope directive takes precedence over the actual parameters so, in
     * this case, needs to be opened independently. */
    (*serialiser)(&tokens[3], NULL, 1, SERIAL_PRECEDED | SERIAL_ENVELOPE_S);

}

/***
* @brief Serialise the records contained within @p set.
*
* The result is similar to this: @verbatim {
* "date" : 'YYYY-MM-DDTHH:mm:ss.000Z',
* "x" : number,
* "y" : number,
* "t" : temperature, // One-digit fraction; in Celsius.
* "ph" : pH, // Not sampled; always 255.
* "rh" : relative humidity // Not sampled; always 255.
* }
@endverbatim
*
* Additional objects may follow.
*
* No flushing is performed.
*
* @param[in,out] set #LogRecordSet from which to extract records. Upon return,
* @p set will be empty unless @p count records have been serialised, instead.
* @param[in] count Maximum number of records to serialise.
* @param[in] is_preceded Whether the records to be serialised are preceded by
* other items; non-zero denotes yes.
* @returns The amount of serialised records.
*/

```

```

static uint8_t rsrc_measurement_serial_log(LogRecordSet* set,
                                         uint8_t count,
                                         uint8_t is_preceded) {
    uint8_t i = 0; /* Counts the amount of serialised records. */
    uint8_t token_buf[47]; /* Key tokens. */
    uint8_t* tokens[6]; /* Pointers to each token in @c token_buf. */

    /* Parameter value buffers. */
    uint8_t s_date [PRM_DATE_LEN]; /* String form of rec.date */
    uint8_t s_temp [PRM_TEMP_LEN]; /* String form of rec.t */

    LogRecord rec;

    /* Set-up output value references. */
    ParamValue params[] = {PARAM_STRING(s_date, PRM_DATE_LEN),
                           PARAM_UINT8(rec.x),
                           PARAM_UINT8(rec.y),
                           PARAM_STRING(s_temp, PRM_TEMP_LEN),
                           PARAM_UINT8(rec.ph),
                           PARAM_UINT8(rec.rh)};

    /* Load tokens into main memory. */
    pgm_read_str_array(tokens, token_buf, prm_date,
                       prm_x,
                       prm_y,
                       prm_t,
                       prm_ph,
                       prm_rh, NULL);

    /* The first record is not preceded by another record. */
    uint8_t serial = SERIAL_ATOMIC_S | SERIAL_ATOMIC_E;
    if(is_preceded) serial |= SERIAL_PRECEDED;

    while(i < count && !log_get_next(&rec, set)) {

        date_to_str(s_date, &rec.date);
        temp_to_str(s_temp, PRM_TEMP_LEN, rec.t);

        (*serialiser)(tokens, params, 6, serial);

        serial |= SERIAL_PRECEDED;
        ++i;
    }
    return i;
}

/**
 * @brief Serialise log records in chunks.

```

```

*
*
* @param[in,out] Set of records to serialise.
* @param[in] page_size The size of each result page.
* @param[in] page_index The index of result page.
* @param[in] total The total amount of available records.
* @param[in] count The amount of records to return (either @p total or @p
* page_size).
*/
static inline void rsrc_measurement_chunk_log(LogRecordSet* set,
                                              uint8_t page_size,
                                              uint8_t page_index,
                                              uint8_t total,
                                              uint8_t count) {

    uint16_t size; /* Size (octets) of each chunk. */
    uint8_t is_next = 0; /* @c 1 for the second group and forth. */
    uint8_t chunk; /* Number of records within this chunk group. */

    /* The preamble accounts for 67 bytes (including envelope start). Serialise
     * a chunk containing statistical info. */
    size = 67;
    srvr_prep_chunk_head(size);
    rsrc_measurement_serial_info(page_index, page_size, total);
    srvr_send(TXF_ln);

    /* Serialise records in groups that fit within the allocated output buffer
     * of the network module. */
    while(count) {
        chunk = count < HTTP_BUF_SIZE/102 ? count : HTTP_BUF_SIZE/102;

        size = 102*chunk - (!is_next);
        srvr_prep_chunk_head(size);
        rsrc_measurement_serial_log(set, chunk, is_next);
        srvr_send(TXF_ln);

        is_next = 1;
        count -= chunk;
    }

    /* rsrc_measurement_info() opens an envelope. Finalise it, as well
     * as the whole object and then flush the response. */
    size = 4;
    srvr_prep_chunk_head(size);

    (*serialiser)(NULL, NULL, 1, SERIAL_ENVELOPE_E
                  | SERIAL_ATOMIC_E
                  | SERIAL_FLUSH);
}

```

```
    srvr_prep(TXF_ln);

    /* Last chunk (should be 0-length). */
    srvr_prep_chunk_head(0);
    srvr_send(TXF_ln);
}

/***
 * @ingroup resource
 * @brief Manage device measurements.
 *
 * Method GET:
 * Fetch logged measurements. By default, all measurements are returned. The
 * following query string parameters may be used to return a subset of results:
 * @verbatim
     date-since // Format ISO8601: YYYY-MM-DDTHH:mm:ss[.sss][Z]
     date-until // Format ISO8601: YYYY-MM-DDTHH:mm:ss[.sss][Z]
     page-index // 0 up to 255
     page-size // 0 up to 255
@endverbatim
 * Additional notes:
 * - Fraction of second and time-zone (in dates) are never parsed and may be
 * omitted. Time-zone is always UTC (@c Z).
 * - Dates since 2000-01-01 up to 2099-12-31 are accepted.
 * - Pagination is applied last, ie, after a subset of results has been
 * selected based on the other parameters, if such parameters were
 * specified.
 * - Any of the above specified parameters may be used, the order of which is
 * irrelevant. If found multiple times, only the first non-empty value of
 * each is preserved.
 * - Unrecognised parameters are ignored. Permissible parameters, although
 * optional, must be valid. Otherwise, 400 is returned.
 * - @c page-index is only relevant, if @c page-size has been specified (to a
 * value greater than @c 0).
 * - To query the amount of records without fetching them, @c page-size may be
 * set to @c 0. @c total will contain the amount of available records (see
 * response format below).
 *
 * Returns:
 * - 200 OK; the body contains the logged measurements. Format: @verbatim {
     "page-index": number,
     "page-size" : number,
     "total" : number,
     "log" : [
         "date" : "YYYY-MM-DDTHH:mm:ss.000Z",
         "x" : number,
         "y" : number,
```

```

    "t" : temperature, // One-digit fraction; in Celsius.
    "ph" : pH, // Not sampled; always 255.
    "rh" : relative humidity // Not sampled; always 255.
}, ...

}] } @endverbatim
* Additional notes:
* — @c page-index is the requested result page. Defaults to @c 0, if none
* was specified.
* — @c page-size is the maximum size of @c log. Defaults to @c total, if
* none was specified.
* — @c total is the available amount of records, if pagination options
* were not applied.
* — @c log contains a maximum of @c page-size measurement records. It is
* always present, even if it is empty.
* — 400 Bad Request; if a wrong value for any of the permissible parameters
* has been specified.
* — 414 Request-URI Too Long; the query string exceeds the allocated buffer
* size. The request should be reconstructed to contain a smaller query
* string.
*
* Example: @verbatim
/measurement?page-size=10&page-index=2&date-since=2015-01-01T12:30:00
@endverbatim
*
* Method POST:
* Performs and records a new measurement.
* The message body should be empty. Returns:
* — 202 Accepted; sampling has been initiated. Header ‘Retry-After’ designates
* the estimated time until completion.
* — 503 Service Unavailable; the device is currently busy (repositioning
* itself or taking a measurement). The request should be reattempted
* later, as specified in the ‘Retry-After’ header.
*/
void rsrc_handle_measurement(HTTPRequest* req) {
    uint8_t status; /* Status of response. */
    uint16_t eta; /* Estimated time until motors complete. */

    if(req->method == METHOD_POST) {

        Position pos;
        if(motor_get(&pos)) {
            status = TXF_STATUS_503;
            eta = task_get_estimate();

        } else {
            task_log_sample(&pos);
            status = TXF_STATUS_202;
        }
    }
}

```

```
    eta = task_get_estimate();
}

} else if(req->method == METHOD_GET) {

    /* Default date limits. */
    BCDDate since = {.year = 0x00, .mon = 0x01, .date = 0x01,
                      .hour = 0x00, .min = 0x00, .sec = 0x00};
    BCDDate until = {.year = 0x99, .mon = 0x12, .date = 0x31,
                      .hour = 0x23, .min = 0x59, .sec = 0x59};

    QueryString* q = &req->query; /* Access to query parameters. */
    uint8_t page_index = 0; /* Requested page index. */
    uint8_t page_size = 0; /* Requested page size. */
    uint8_t total; /* Total available records. */
    uint8_t count; /* Amount of records returned. */

    uint8_t is_size = 0; /* Flags whether page-size was set. */
    uint8_t errors = 0; /* Parser errors. */
    uint16_t size = 71; /* Content-length with 0 records. */

    LogRecordSet set; /* Results that match current params.*/

    /* Parse string values for the query string. */
    if(q->values[PRM_MSR_DATE_SINCE]) {
        errors += str_to_date(&since, q->values[PRM_MSR_DATE_SINCE]);
    }
    if(q->values[PRM_MSR_DATE_UNTIL]) {
        errors += str_to_date(&until, q->values[PRM_MSR_DATE_UNTIL]);
    }
    if(is_size = (q->values[PRM_MSR_PAGE_SIZE] != 0)) {
        page_size = atoi(q->values[PRM_MSR_PAGE_SIZE]);

        /* Page index is relevant only if page size has been specified. */
        if(q->values[PRM_MSR_PAGE_INDEX]) {
            page_index = atoi(q->values[PRM_MSR_PAGE_INDEX]);
        }
    }
}

/* Execute the request, if there were no errors in the params.*/
if(!errors) {

    /* Find records within the specified dates. */
    total = log_get_set(&set, &since, &until);

    if(is_size) {
        /* Skip the specified amount of records. @c count may be less or
         * equal to @c total. */
    }
}
```

```

    count = log_skip(&set, page_size*page_index);

    /* Return the lesser amount of records between @c page_size and
     * @c count. */
    if(page_size < count) count = page_size;

} else {
    /* All the records are returned in a single page which contains
     * @c total records. */
    count = total;
}

if(count) {
    /* @c 102 includes the object separator (comma) for @c count
     * records (the actual size of each being 101 bytes), which is
     * not incorporated with the first record (and thus, @c -1). */
    size += 102*count - 1;
}

/* Serialise in chunks. */
srvr_prep(TXF_STATUS_200, TXF_ln,
          TXF_STANDARD_HEADERS_ln,
          TXF_CONTENT_TYPE_JSON_ln,
          TXF_CACHE_NO_CACHE_ln,
          TXF_CHUNKED,
          TXF_lnl);

rsrc_measurement_chunk_log(&set,
                           page_size,
                           page_index,
                           total,
                           count);

/* Return 400 on erroneous parameter values. */
} else {
    status = TXF_STATUS_400;
}
}

switch(status) {
    case TXF_STATUS_202:
        srvr_send(TXF_STATUS_202, TXF_ln,
                  TXF_STANDARD_HEADERS_ln,
                  TXF_CACHE_NO_CACHE_ln,
                  TXF_CONTENT_LENGTH_ZERO_ln,
                  TXF_RETRY_AFTER, TXF_HS, TXFx_FW_UINT, eta,
                  TXF_lnl);
}

```

```

break;
case TXF_STATUS_400:
    srvr_send(TXF_STATUS_400, TXF_ln,
               TXF_STANDARD_HEADERS_ln,
               TXF_CACHE_NO_CACHE_ln,
               TXF_CONTENT_LENGTH_ZERO_ln, TXF_ln);
break;
case TXF_STATUS_503:
    srvr_send(TXF_STATUS_503, TXF_ln,
               TXF_STANDARD_HEADERS_ln,
               TXF_CACHE_NO_CACHE_ln,
               TXF_CONTENT_LENGTH_ZERO_ln,
               TXF_RETRY_AFTER, TXF_HS, TXFx_FW_UINT, eta,
               TXF_lnl);
break;
/* Case TXF_STATUS_200 is implemented in-line, above. */
}
}

```

### Διεπαφή ολοκληρωμένου πραγματικού χρόνου (RTC)

#### rtc.h

```

/**
* @file
*/
#ifndef RTC_H_INCL
#define RTC_H_INCL

#include <inttypes.h>

/**
* @brief Structure of the DS1307 RTC memory map.
*
* These registers are both readable and writable and internally buffered. All
* values are in BCD format. *DS1307 p.8*
*/
typedef struct {

    /** @brief Seconds elapsed (00–59).
    *
    * Bit @c 7 corresponds to #RTC_CH.
    */
    uint8_t sec;

    /** @brief Minutes elapsed (00–59). *DS1307 p.8* */
    uint8_t min;
}

```

```

/** @brief Hours elapsed (01–12+AM/PM or 00–23).
 *
 * Bit @c 6 corresponds to #RTC_HMODE; bit @c to #RTC_AM_PM.
 */
uint8_t hour;

/** @brief Day of week (01–07).
 *
 * Any day may be specified as the first day of the week. *DS1307 p.8*
 */
uint8_t day;

/** @brief Date (01–31). *DS1307 p.8* */
uint8_t date;

/** @brief Month (01–12). *DS1307 p.8* */
uint8_t mon;

/** @brief Year (00–99). *DS1307 p.8* */
uint8_t year;

/** @brief Controls the output on pin @c SQW/OUT.
 *
 * — Bit @c 7 corresponds to bit #RTC_SQW_OUT.
 * — Bit @c 4 corresponds to bit #RTC_SQWE.
 * — Bits @c 1 and @c 0 correspond to #RTC_RS1 and #RTC_RS0, respectively.
 */
uint8_t sqw;
} RTCMap;

/** @brief Address of DS1307 on TWI bus.
 *
 * The address of the DS1307 is @c 1101000 (*DS1307 p.12*). When the address (7
 * bits long) is transmitted over the bus, it is followed by the operation bit
 * (@c\f$R/\overline{W}\f$). The address is shifted one time to the left so it
 * can facilitate adding the operation bit on the LSB of that data byte.
 */
#define RTC_ADDR (0xDO)

/** @brief Enables or disables the timekeeping operation (Clock Halt — CH).
 *
 * Setting this bit results in stopping the RTC and consumption is further
 * reduced. Clearing it enables the oscillator. Upon first power application, it
 * defaults to @c 1. *DS1307 p.8*
 */

```

```
#define RTC_CH 7

/**
 * @brief Chooses between 12- and 24-hour mode.
 *
 * @c 1 is for 12-hour mode in which case, bit #RTC_AM_PM corresponds to AM/PM;
 * otherwise, it is part of the ten-hours BCD digit. *DS1307 p.8*
 */
#define RTC_HMODE 6

/**
 * @brief AM/PM or ten-hours of BCD digit.
 *
 * If bit @c RTC_HMODE is set, then 12-hour mode of operation is selected; this
 * bit corresponds to AM/PM, @c 0 denoting AM. If in 24-hour mode, this bit is
 * part of BCD hour value.
 */
#define RTC_AM_PM 5

/**
 * @brief Determines the constant output on pin @c SQW/OUT.
 *
 * When bit #RTC_SQWE is cleared (ie, the square-wave output is disabled),
 * if this bit is cleared as well, @c SQW/OUT is set to low; otherwise, it is set
 * to high.
 */
#define RTC_SQW_OUT 7

/**
 * @brief Enables or disables waveform generation.
 *
 * If set, a square-wave is generated on pin @c SQW/OUT of the frequency
 * determined by bits #RTC_RS1 and #RTC_RSO.
 */
#define RTC_SQWE 4

/**
 * @brief Determines, along with #RTC_RSO, the frequency of the square-wave
 * output.
 *
 * For details, see #RTC_RS0.
 */
#define RTC_RS1 1

/**
 * @brief Determines, along with #RTC_RS1, the frequency of the square-wave
 * output.
 *
```

```

*
* To enable the square-wave output on pin @c SQW/OUT, bit #RTC_SQWE must be set.
* The value of RTC_RS[1:0] control the generated wave frequency as follows:
* — @c 0 outputs 1Hz.
* — @c 1 outputs 4.096kHz.
* — @c 2 outputs 8.192kHz.
* — @c 3 outputs 32.768kHz.
*/
#define RTC_RS0 0

/**
* @brief Set the RTC time.
*
* It is a convenience wrapper around rtc_write().
*
* @param[in] rtc The bytes to send to the RTC. If @c -1 is returned, not all
* bytes may have been sent to the RTC.
* @returns @c 0 on success; @c -1, otherwise.
*/
int8_t rtc_set(RTCMap* rtc);

/**
* @brief Get the RTC time.
*
* Updates the members of @p rtc to reflect the RTC values (first 8 Bytes). It is
* a convenience wrapper around rtc_read().
*
* @param[out] rtc The bytes received from the RTC. If @c -1 is returned, the
* contents of @p rtc may be partially updated.
* @returns @c 0 on success; @c -1, otherwise.
*/
int8_t rtc_get(RTCMap* rtc);

/**
* @brief Write a number of bytes to the RTC memory.
*
* @param[in] addr The RTC word address to start writing to (@c 0---@c 3F).
* @param[out] buf An array of bytes to send to the RTC.
* @param[in] len The number of bytes to send.
* @returns @c 0 on success; @c -1, otherwise.
*/
int8_t rtc_write(uint8_t addr, uint8_t* buf, uint8_t len);

/**
* @brief Read a number of bytes from the RTC memory.
*
* @param[in] addr The RTC word address to start reading from (@c 0---@c 3F).
* @param[out] buf A pre-allocated array to write the bytes read from the RTC

```

```
* memory into.  
* @param[in] len The number of bytes to read. @p buf should be at least this  
* large.  
* @returns @c 0 on success; @c -1, otherwise.  
*/  
int8_t rtc_read(uint8_t addr, uint8_t* buf, uint8_t len);  
  
/**  
* @brief Set the DS1307 register pointer to the specified address.  
*  
* The DS1307 utilizes an internal register pointer for all read and write  
* operations so that each time a byte is sent or received, the pointer is  
* automatically incremented to point to the next memory address (*DS1307 p.12*).  
* This function helps to set the initial value of that register pointer.  
*  
* Note, this function does not release the TWI bus upon successful pointer  
* initialization but only in the event of a failure.  
*  
* @returns @c 0 on success; @c -1, otherwise.  
*/  
static int8_t rtc_set_pointer(uint8_t addr);  
  
#endif /* RTC_H_INCL */
```

**rtc.c**

```
#include "rtc.h"  
#include "defs.h"  
#include "twi.h"  
  
#include <avr/io.h>  
#include <inttypes.h>  
  
int8_t rtc_set(RTCMap* rtc) {  
    return rtc_write(0, (uint8_t*)rtc, sizeof (RTCMap));  
}  
  
int8_t rtc_get(RTCMap* rtc) {  
    return rtc_read(0, (uint8_t*)rtc, sizeof (RTCMap));  
}  
  
int8_t rtc_write(uint8_t addr, uint8_t* buf, uint8_t len) {  
    uint8_t i = 0;  
  
    /* Set register pointer to the appropriate word address. */  
    if(rtc_set_pointer(addr)) return -1;  
  
    if(!len) return 0;
```

```

/* Send the new values, byte-after-byte. */
do {
    TWDR = buf[i];
    TWI_DO_WAIT();
    ++i;
} while(i < len && TWI_STATUS() == TWI_DATA_W_ACK);

/* Notify end-of-transmission. */
TWI_STOP();

/* If an error has occurred, report it. */
if(TWI_STATUS() != TWI_DATA_W_ACK) return -1;

return 0;
}

int8_t rtc_read(uint8_t addr, uint8_t* buf, uint8_t len) {
    uint8_t i = 0;
    uint8_t byte;

    /* Set DS1307 pointer to @p addr; the word address to read. */
    if(rtc_set_pointer(addr)) return -1;

    /* Then, send a repeated start and read the desired length of bytes. The
     * last byte to be read should be followed by a NACK. *DS1307 p.12* */
    TWI_ATTEMPT(TWI_START(), TWI_RSTART);

    /* Select RTC in read mode. */
    TWI_ATTEMPT(TWI_SLA_R(RTC_ADDR), TWI_SLA_R_ACK);

    /* Read @p len bytes acknowledging each except for the last one. */
    do {
        TWI_DO_ACK();
        TWI_ATTEMPT(TWI_WAIT(), TWI_DATA_R_ACK);

        byte = TWDR;
        buf[i] = byte;

        ++i;
    } while(i < len - 1);

    /* Read the last byte of @p rtc without acknowledging it. *Atmel p.224*,
     * *DS1307 p.10* */
    TWI_ATTEMPT(TWI_DO_WAIT(), TWI_DATA_R_NACK);

    /* Load the last byte into @p rtc. */
    byte = TWDR;
}

```

```

buf[i] = byte;

/* Release the bus. */
TWI_STOP();
return 0;
}

static int8_t rtc_set_pointer(uint8_t addr) {
    /* Make sure to re-initialise the internal TWI state machine after a
     * possible power-down. */
    TWI_INIT();

    /* Send start condition. */
    TWI_ATTEMPT(TWI_START(), TWI_SSTART);

    /* Select RTC in write mode. */
    TWI_ATTEMPT(TWI_SLA_W(RTC_ADDR), TWI_SLA_W_ACK);

    /* Transmit the DS1307 register to start operating upon. */
    TWDR = addr;
    TWI_DO_WAIT();

    if(TWI_STATUS() != TWI_DATA_W_ACK) {
        TWI_STOP();
        return -1;
    }

    return 0;
}

```

## Poή χαρακτήρων Socket

### sbuffer.h

```

/**
 * @file
 * @addtogroup sbuffer Socket Buffer
 * @ingroup http_server
 * @{
 */

#ifndef SBUFFER_H_INCL
#define SBUFFER_H_INCL

#include "defs.h"
#include <inttypes.h>

/**
 * @brief Specifies which socket's data are to be buffered locally in the MCU.

```

```

*
* Functions s_next(), s_peek() and s_drop() operate on a fragment of the network
* module's data, internally buffered in the SRAM of the MCU. The size of this
* buffer, which is determined by #NET_BUF_LEN and is, typically, (much) smaller
* than the size of the module's input buffer, will need to be updated with
* successive fragments (for details, see s_update()). Updating the internal
* buffer is handled by these functions transparently.
*
* In order to facilitate internal input buffering for any socket (any one at a
* time), the socket the data of which are to be buffered needs to be specified
* using this function before attempting any operations with the aforementioned
* functions (or derivatives, thereof). It is not necessary to call this before
* any other function, but only when switching input from a different socket.
*
* @param[in] s Socket to buffer data from.
*/
void set_socket_buf(uint8_t s);

/**
* @brief Read the next byte from the network input stream.
*
* This function actually reads the next byte from the internal buffer. It causes
* a buffer update (see s_update()) when all bytes have been depleted and a new
* one is requested.
*
* #EOF is returned when no bytes are available even after requesting a buffer
* update.
*
* @param[out] c The byte read from stream.
* @returns 0 on success; #EOF on end-of-stream.
*/
int8_t s_next(uint8_t* c);

/**
* @brief Read into @p c the byte at offset @p pos from the current position in
* the stream.
*
* This function does not consume bytes from the stream, ie, the next byte read
* by s_next() is the same as if s_peek() was not invoked at all. It does cause a
* buffer update, if not enough bytes are present in the buffer to perform the
* operation.
*
* @param[out] c The byte read at offset @p pos.
* @param[in] pos The offset to read a byte from.
* @returns 0 on success; #EOF on end-of-stream.
*/
int8_t s_peek(uint8_t* c, uint16_t pos);

```

```

/** 
 * @brief Discard @p count bytes from the stream.
 *
 * It causes a buffer update, if not enough bytes are present in the buffer to
 * perform the operation. If, even a buffer update, the operation cannot be
 * completed at its entirety, it is aborted altogether (ie, no bytes are
 * discarded).
 *
 * @param[in] count The amount of bytes to discard.
 * @returns 0 on success; #EOF on end-of-stream.
 */
int8_t s_drop(uint16_t count);

/** 
 * @brief Update the contents of the internal network input buffer.
 *
 * When this function is invoked, a fragment of the available data on the socket
 * specified by set_socket_buf() are read into the internal buffer. The size of
 * the fragment is the least amount of bytes between the bytes available on the
 * socket and the available space (ie, consumed bytes) of the internal buffer.
 *
 * If s_update() is invoked when no data are available on the socket, #EOF is
 * returned and the internal buffer remains intact. If s_update() is invoked when
 * the internal buffer is full, the function completes successfully although no
 * new bytes are actually loaded.
 *
 * To set the size of the internal input buffer, refer to #NET_BUF_LEN. To set
 * input buffering for a particular socket, refer to set_socket_buf().
 *
 * @returns @c 0 if new data were available; #EOF on end-of-stream.
 */
static int8_t s_update();

#endif /* SBUFFER_H_INCL */
/** @}

```

**sbuffer.c**

```

#include "sbuffer.h"
#include "w5100.h"
#include <stdio.h>

/** 
 * @ingroup sbuffer
 * @brief The internal input buffer.
 *
 * Maintains a fragment of the available network data for immediate access.
 */

```

```

static uint8_t buf[NET_BUF_LEN];

/**
 * @ingroup sbuffer
 * @brief The amount of valid data in #buf.
 */
static uint16_t buf_data = 0;

/**
 * @ingroup sbuffer
 * @brief The offset within #buf of the next-to-read byte.
 */
static uint16_t buf_RD = 0;

/**
 * @ingroup sbuffer
 * @brief The offset within #buf of the next-to-write byte.
 */
static uint16_t buf_WR = 0;

/**
 * @ingroup sbuffer
 * @brief The socket to buffer data from.
 */
static uint8_t buf_Sn = 0;

void set_socket_buf(uint8_t s) {
    buf_RD = 0;
    buf_WR = 0;
    buf_Sn = s;
}

int8_t s_next(uint8_t* c) {
    /* When the two offsets point at the same buffer position, then all valid
     * data has been read and more are needed to be loaded. */
    if(buf_data == 0) {
        s_update();
    }

    /* If there are data loaded into @c buf, */
    if(buf_data > 0) {
        *c = buf[buf_RD];
        ++buf_RD;
        --buf_data;
        if(buf_RD == NET_BUF_LEN) buf_RD = 0;

        /* If, after an update request, data are not available then there is nothing
         * more to load from W5100. */
    }
}

```

```
    } else {
        return EOF;
    }
    return 0;
}

int8_t s_peek(uint8_t* c, uint16_t pos) {
    /* If the offset from @c buf_RD (@p pos) exceeds the amount of available
     * bytes then more must be loaded into the local memory first. */
    if(pos >= buf_data) {
        s_update();
    }

    /* Peak forward only if there are enough data loaded into @c buf. */
    if(pos < buf_data) {

        /* Return byte at @p pos when there are enough bytes between @c buf_RD
         * and @c buf_WR or the end of the buffer. */
        if(buf_RD < buf_WR || buf_RD + pos < NET_BUF_LEN) {
            *c = buf[buf_RD + pos];

        } else {
            *c = buf[pos - (NET_BUF_LEN - buf_RD)];
        }

        /* If, after an update request, data are not enough then there is nothing
         * at position @p pos. */
    } else {
        return EOF;
    }

    return 0;
}

int8_t s_drop(uint16_t count) {
    /* If the offset from @c buf_RD (@p pos) exceeds the amount of available
     * bytes then more must be loaded into the local memory first. */
    if(count > buf_data) {
        s_update();
    }

    /* Drop the requested amount of bytes only if there are enough loaded into
     * @c buf. */
    if(count <= buf_data) {

        /* Return byte at @p pos when there are enough bytes between @c buf_RD
         * and @c buf_WR or the end of the buffer. */
        if(buf_RD < buf_WR || buf_RD + count <= NET_BUF_LEN) {
```

```

        buf_RD += count;

    } else {
        buf_RD = buf[count - (NET_BUF_LEN - buf_RD)];
    }

    buf_data -= count;

/* If, after an update request, data are not enough then there is nothing
 * at position @p pos. */
} else {
    return EOF;
}

return 0;
}

static int8_t s_update() {
    uint16_t rx_size = net_read16(NET_Sn_RX_RSR(buf_Sn)); /* Available data. */
    uint16_t fragment; /* Actual amount of bytes to be read. */

    /* Read a chunk from the available data up to a maximum of @c NET_BUF_SIZE
     * and fit it into @c buf. */
    if(rx_size > 0) {

        /* @c buf may not be completely empty at the time of invocation. That
         * amount of data is represented by @c buf_data and conserve space until
         * read (ie, @buf_RD is promoted over them). */
        if(rx_size <= NET_BUF_LEN - buf_data) {
            fragment = rx_size;
        } else {
            fragment = NET_BUF_LEN - buf_data;
        }

        /* Depending on the state of the RD/WR offsets, data may not be able to
         * be written in one contiguous block but, rather, wrapped around the end
         * of the buffer. */
        if(buf_WR + fragment > NET_BUF_LEN) {
            uint16_t bound = NET_BUF_LEN - buf_WR;

            net_recv(buf_Sn, &(buf[buf_WR]), bound); /* Read up to buffer limit. */

            /* Read the remainder of bytes. */
            net_recv(buf_Sn, buf, fragment - bound);
            buf_WR = fragment - bound; /* Update WR offset. */
        } else {
            net_recv(buf_Sn, &(buf[buf_WR]), fragment);
            buf_WR += fragment; /* Update WR offset. */
        }
    }
}

```

```

    }

    buf_data += fragment; /* Update the amount of in-buffer data. */
} else {
    return EOF;
}
return 0;
}

```

## Αισθητήρες υλοποίησης

### sensor.h

```

/** 
 * @file
 */

#ifndef SENSOR_H_INCL
#define SENSOR_H_INCL

#include "defs.h"

#include <inttypes.h>

/** 
 * @brief Activates access to the 1-wire DQ line.
 *
 * Sets the appropriate signals so that the 1-wire DQ line may be access through
 * #MUX_2Z pin. It also enables the multiplexer it is connected to.
 *
 * Note that the 1-wire DQ may not be access at the same time the motors are
 * being operated.
 */
#define W1_ENABLE() \
MUX_SO_PORT |= _BV(MUX_SO); \
MUX_S1_PORT |= _BV(MUX_S1); \
MUX_ENABLE()

/** 
 * @brief Disable access to the 1-wire DQ line.
 *
 * This actually disables the multiplexer. It is advised the DQ line always be
 * deactivated this way because other components that share the multiplexer may
 * expect it to be disabled by default.
 */
#define W1_DISABLE() \
MUX_DISABLE()

/**

```

```

* @brief Samples the DS18B20 digital thermometer and returns its reading.
*
* This function expects the DS18B20 to be the only slave connected to the 1-wire
* DQ line. It returns the reading in a 16-bit, sign-extended two's complement
* format, as follows:
* — Bits @c 15—11 contain the sign.
* — Bits @c 10—4 the integral part of the reading (@f$2^{\{6\}}—2^{\{0\}}@f$).
* — Bits @c 3—0 the fraction of the reading (@f$2^{\{-1\}}—2^{\{-4\}}@f$).
*
* @returns The temperature reading of the sensor.
*/
uint16_t sens_read_t();

#endif /* SENSOR_H_INCL */

sensor.c

#include "sensor.h"
#include "onewire.h"

uint16_t sens_read_t() {
    uint16_t t;

    W1_ENABLE();

    /* All transactions on the 1-wire bus begin with an initialisation sequence
     * (*DS18B20 p.11*). Do not proceed if no devices were detected, and in this
     * case, the DS18B20) . */
    if(w1_reset()) return -1;

    /* Currently, we operate in a single drop bus system so skip ROM
     * operations. *DS18B20 p.11* */
    w1_write(W1_ROM_SKIP);
    w1_write(W1_CONVERT_T);

    /* Allow for a 12-bit-resolution conversion to complete. *DS18B20 p.9* */
    _delay_ms(750);

    if(w1_reset()) return -1;

    w1_write(W1_ROM_SKIP);
    w1_write(W1_READ_SCRATCHPAD);
    t = w1_read(12);

    /* Halt DS18B20 operation. */
    w1_reset();

    W1_DISABLE();
    return t;
}

```

}

## Συμπληρωματικές λειτουργίες ροής

### stream\_util.h

```
/**  
 * @file  
 * @addtogroup stream_util Stream Utilities  
 * @ingroup http_server  
 *  
 * @brief Utility functions that operate on an input stream, typically from the  
 * network module.  
 */  
  
#ifndef STREAM_UTIL_H_INCL  
#define STREAM_UTIL_H_INCL  
  
#include <inttypes.h>  
  
/**  
 * @brief Indicates any character (based on the context of its use).  
 */  
#define OTHER -5  
  
#ifndef EOF  
/**  
 * @brief Indicates that the End-of-File has been reached.  
 */  
#define EOF -1  
#endif  
  
/**  
 * @brief Sets the function that supplies this module with bytes from the input  
 * stream.  
 *  
 * Sets the value of #gnext.  
 * The provided function should accept a single byte address into which to store  
 * the next byte found on the stream. It should return @c 0 if the contents of  
 * that byte had been successfully updated or #EOF, if no more bytes are  
 * available on the stream for this particular processing cycle.  
 *  
 * Note that if any of the provided functions of this module are invoked without  
 * previously setting the function pointer will, in all probability, cause the  
 * application to fail.  
 *  
 * @param[in] input_source Pointer to input function.  
 */  
void stream_set_source(int8_t (*input_source)(uint8_t*));
```

```
/**  
 * @brief Read up to a four-digit unsigned hexadecimal number from stream  
 * (0 up to FFFF).  
 *  
 * Any leading zeros are ignored. Prefix "0x" is not supported.  
 *  
 * @param[out] value The number read. Defaults to @c 0.  
 * @param[in,out] c The first character to start parsing from and the last one  
 * read from the stream.  
 * @returns One of:  
 * — @c 0; if the last character read is not a digit or EOF.  
 * — #OTHER; if a greater number than FFFF is available.  
 * — EOF  
 */  
int8_t parse_hex16(uint16_t* value, uint8_t* c);  
  
/**  
 * @brief Read up to a 4-digit unsigned number from stream (0 up to 9999).  
 *  
 * Any leading zeros are ignored.  
 *  
 * @param[out] value The number read. Defaults to @c 0.  
 * @param[in,out] c The first character to start parsing from and the last one  
 * read from the stream.  
 * @returns One of:  
 * — 0; if the last character read is not a digit or EOF.  
 * — #OTHER; if a greater number than 9999 is available.  
 * — EOF  
 */  
int8_t parse_uint16(uint16_t* value, uint8_t* c);  
  
/**  
 * @brief Read up to a two-digit unsigned number from stream (0 up to 99).  
 *  
 * Any leading zeros are ignored.  
 *  
 * @param[out] value The number read. Defaults to @c 0.  
 * @param[in,out] c The first character to start parsing from and the last one  
 * read from the stream.  
 * @returns One of:  
 * — 0; if the last character read is not a digit or EOF.  
 * — #OTHER; if a greater number than 99 is available.  
 * — EOF  
 */  
int8_t parse_uint8(uint8_t* value, uint8_t* c);  
  
/**
```

```
* @brief Copy from the stream into the specified buffer.  
*  
* The process is terminated when @p delim is read from the stream or a total of  
* (@p max - 1) bytes have been copied, whichever occurs first. The last byte  
* inserted into @p buf is always byte @c 0 (null-byte or string-terminator). If  
* the termination occurs due to reaching @p max and not reading the delimiter,  
* #OTHER is returned.  
*  
* @param[out] buf The memory address to start copying bytes into.  
* @param[in] delim The copy terminator, which will be replaced by a null-byte  
* into @p buf.  
* @param[in] max Maximum numbers of bytes (including null-byte) to copy into @p  
* buf, if @c delim is not found first.  
* @param[in,out] c The first byte to start copying from and the last one read  
* from the stream.  
* @returns One of:  
* - @c 0; if @c delim was found without exceeding @c max.  
* - #OTHER; if more bytes are available on the stream, but @p max has been  
* reached.  
* - #EOF; if End-of-stream has been reached while copying bytes.  
*/  
int8_t copy_until(uint8_t* buf, uint8_t delim, uint8_t max, uint8_t* c);  
  
/**  
* @brief Find the closest match from an array of strings with the stream.  
*  
* Accepts an array of strings (descriptors) and compares the against the input  
* stream. The descriptors should be sorted in ascending order beforehand, if  
* necessary, and no duplicates should exist. Otherwise, the result is undefined.  
*  
* The return value is the index of the closest matching descriptor (which may or  
* may not be a valid match -- more on this later); the value #OTHER, if it is  
* certain no descriptor provides a match; or @c EOF if the end of the stream  
* has been reached in the meantime.  
* It should be noted that this function irreversibly consumes bytes from the  
* input stream which are not reinstated back into it even if no match is found.  
*  
* More specifically, on each iteration a single character is extracted from the  
* input stream and is compared against the known descriptors and, in particular,  
* against the character at a position equal to the number of characters read  
* (ie, the first character read is compared against the first character of all  
* descriptors, etc).  
*  
* Initially, all the descriptors are included in the comparisons. However, as  
* the iterations progress, descriptors that fail a match are omitted from the  
* remaining iterations. Because the descriptors are given in ascending order,  
* this is easily done by maintaining two moving boundaries.  
*/
```

```

* By the end of the iterations, the two boundaries have converged and specify
* a particular descriptor. That descriptor is a match if two conditions apply:
* — Its last character has been reached (null-character).
* — The input character that caused that last mismatch is an acceptable
* delimiter for this particular input.
*
* The former is readily determined whilst the latter, not. This is because this
* function is content-unaware and, thus, ignorant of what constitutes an
* acceptable delimiter. Consequently, it cannot fully determine whether the
* index returned actually corresponds to a full match; that is the
* responsibility of the caller. It can, however, determine with certainty when
* there is no match, in which case, it returns #OTHER.
*
* @param[in] desc Array with strings (descriptors).
* @param[in] max The upper boundary of @p desc; index of the last literal to use
* in the comparisons, incremented by 1.
* @param[in,out] c The first character to compare against the strings and the
* last one read from the stream.
* @returns One of:
* — Index of @p desc with a possible match.
* — #OTHER; on certainty of no match.
* — EOF; if end of stream has been reached before hitting a match/mismatch.
*/
int8_t stream_match(uint8_t** desc, uint8_t max, uint8_t* c);

/**
* @brief Find the closest match from an array of strings with the stream.
*
* It provides comparable functionality to stream_match() but with greater
* flexibility as far as the iteration termination is concerned. More
* specifically, new iterations take place as long as there is at least one match
* per iteration. Upon failure, the function returns with neither @p max or @p
* cmp_idx being updated.
* It is the caller's responsibility to determine whether the iterations should
* proceed as well as whether they should do so from where they had previously
* terminated or even with which character.
*
* stream_match() could actually be implemented as a wrapper around this one.
*
* @param[in] desc Array with strings (descriptors).
* @param[in] abs_min Index of the first acceptable element of the array. It is
* different than @p min in that this one represents the absolute minimum value
* of the array (below which there is no valid descriptor).
* @param[in,out] min The lower boundary of matches in @p desc. Upon first
* invocation, this should be equal to @p abs_min. As iterations take place,
* this may be increased internally.
* @param[in,out] max The upper boundary of @p desc. Upon first invocation, this
* should be equal to the index of the last descriptor plus 1.

```

```

* @param[in,out] cmp_idx Character index of strings found within @p desc that
* is to be compared against @p c and the rest of the stream. Upon first
* invocation, this should be 0.
* @param[in,out] c The first character to compare against the strings and the
* last one read from the stream.
* @returns One of:
* — Index of @p desc with a possible match.
* — #OTHER; on certainty of no match.
* — EOF; if end of stream has been reached before hitting a match/mismatch.
*/
int8_t stream_match_ext(uint8_t** desc,
                       uint8_t abs_min,
                       uint8_t* min,
                       uint8_t* max,
                       uint8_t* cmp_idx,
                       uint8_t* c);

#endif /* STREAM_UTIL_H_INCL */

stream_util.c

#include "stream_util.h"

/**
* @ingroup stream_util
* @brief Function pointer to access the next character to parse.
*
* It provides this module's components access to the input stream. It is set
* using stream_set_source(). Attempting to invoke any of the functions included
* in this module without previously setting it to a known value will, in all
* probability, cause the application to fail.
*/
static int8_t (*gnext)(uint8_t*);

void stream_set_source(int8_t (*next_char)(uint8_t*)) {
    gnext = next_char;
}

int8_t parse_hex16(uint16_t* value, uint8_t* c) {
    int8_t c_type = 0;
    int8_t digits;
    *value = 0;

    while(*c == '0') c_type = (*gnext)(c); /* Ignore any leading zeros. */

    for(digits = 0 ; c_type != EOF && digits < 4 && isxdigit(*c) ; ++digits) {

        *value *= 16;
        *value += *c <= '9' ? *c - '0' : tolower(*c) - 'a' + 10;
    }
}

```

```

    c_type = (*gnext)(c);
}

if(isxdigit(*c)) {
    c_type = OTHER; /* Inform an additional digit is on stream. */
}
return c_type;
}

int8_t parse_uint16(uint16_t* value, uint8_t* c) {
    int8_t c_type = 0;
    int8_t digits;
    *value = 0;

    while(*c == '0') c_type = (*gnext)(c); /* Ignore any leading zeros. */

    for(digits = 0 ; c_type != EOF && digits < 4 && isdigit(*c) ; ++digits) {
        *value *= 10;
        *value += *c - '0';
        c_type = (*gnext)(c);
    }
    if(isdigit(*c)) {
        c_type = OTHER; /* Inform an additional digit is on stream. */
    }
    return c_type;
}

int8_t parse_uint8(uint8_t* value, uint8_t* c) {
    int8_t c_type = OTHER;
    int8_t digits;
    *value = 0;

    while(*c == '0') c_type = (*gnext)(c); /* Ignore any leading zeros. */

    for(digits = 0 ; c_type != EOF && digits < 2 && isdigit(*c) ; ++digits) {
        *value *= 10;
        *value += *c - '0';
        c_type = (*gnext)(c);
    }
    if(isdigit(*c)) {
        c_type = OTHER; /* Inform an additional digit is on stream. */
    }
    return c_type;
}

int8_t copy_until(uint8_t* buf, uint8_t delim, uint8_t max, uint8_t* c) {
    int8_t c_type = 0;
    uint8_t i = 0;

```

```
while(*c != delim && !c_type) {
    if(i == max - 1) {
        c_type == OTHER;
        break;
    }

    buf[i] = *c;
    c_type = (*gnext)(c);
    ++i;
}

/* Append a string delimiter. */
buf[i] = 0;
return c_type;
}

int8_t stream_match(uint8_t** desc, uint8_t max, uint8_t* c) {
    int8_t c_type = 0;
    uint8_t cmp_idx = 0;
    uint8_t i;
    uint8_t min = 0;

    /* When @c min is equal to @c max, the descriptor at position max-1 is a
     * possible match. What is left is to check whether the last character read
     * corresponds to an appropriate delimiter for this particular operation. */
    while(!c_type && min < max) {

        /* The last byte of the descriptors (null-character) is used as an
         * implicit iteration terminator when all the previous characters have
         * been matched. If a null-character is read from the stream when a
         * comparison with the last character of a matching descriptor is due,
         * the comparison will succeed and a new iteration will take place
         * during which the limits of the descriptor will be breached. */
        if(*c == '\0') *c = 1;

        *c = tolower(*c);

        /* Omit descriptors with a character less than @c c. */
        for(i = min; i < max && desc[i][cmp_idx] < *c; ++i)
            ;
        min = i;

        /* Determine position of last possible match. */
        for(i; i < max && desc[i][cmp_idx] == *c ; ++i)
            ;
        max = i;
    }
}
```

```

/* Avoid reading the next character for stream, if lower and upper limit
 * have converged. */
if(min < max) {
    ++cmp_idx;
    c_type = (*gnext)(c);
}
}

/* If there's been an error reading from stream, return that error.
 * Currently, @c EOF is the only possible error. */
if(c_type) return c_type;

/* @c i equals @c 0 when the input string alphabetically precedes the first
 * currently available. */
if(i > 0 && *c != '\0') {
    if(desc[i - 1][cmp_idx] == '\0') return i - 1;
}

return OTHER;
}

int8_t stream_match_ext(uint8_t** desc,
                       uint8_t abs_min,
                       uint8_t* min,
                       uint8_t* max,
                       uint8_t* cmp_idx,
                       uint8_t* c) {
uint8_t i = 0;
int8_t c_type = 0;
int8_t have_hit = 1;

/* The iterations continue as long as there is at least one match. */
while(!c_type && have_hit) {

    have_hit = 0;
    /* Null-character is used implicitly as a comparison terminator. */
    if(*c == '\0') *c = 1;

    *c = tolower(*c);

    /* Omit descriptors with a character less than @c c. */
    for(i = *min; i < *max && desc[i][*cmp_idx] < *c; ++i)
        ;
    *min = i;

    /* Determine position of last possible match. */
    for(i; i < *max && desc[i][*cmp_idx] == *c ; ++i) {
        have_hit = 1;
    }
}
}

```

```

}

/* Lower upper-bound only if there had been hits. */
if(have_hit) *max = i;

/* Read next character if the current one provides a match. Otherwise,
 * the caller should determine if an alternative character should be used
 * in its place. */
if(*min < *max && have_hit) {
    ++(*cmp_idx);
    c_type = (*gnext)(c);
}
}

/* If there's been an error reading from stream, return that error. */
if(c_type) return c_type;

/* @c i equals 'abs_min' when the input string alphabetically precedes the
 * first possible literal. */
if(i > abs_min) {
    if(desc[i - 1][*cmp_idx] == '\0') return i - 1;
}

return OTHER;
}

```

## Εργασία μετρήσεων

### task.h

```

/**
 * @file
 * @addtogroup task Task
 */

#ifndef TASK_H_INCL
#define TASK_H_INCL

#include "defs.h"

#include <inttypes.h>

/**
 * @brief Settings of automated tasks.
 */
typedef struct {
    /**
     * The time elapsed between successive automated samplings. The range is @c
     * 0--@c 240, with @c 0 being none. Each unit describes 6 minutes. So, a

```

```

    * value of @c 60 implies 10 @c samples per hour.
    */
    uint8_t interval;

    /** @brief The amount of samples to take after each @c interval. */
    uint8_t samples;
} Task;

/** @brief Maximum value for Task#interval.
 */
#define TASK_INTERVAL_MAX 240

/** @brief Estimate for how long it takes to get to a new position.
 *
 * It is used to estimate how long it will take to get to each new X-Y coordinate
 * generated by make_target() when #pending_samples is greater than @c 1. This
 * estimate should not include the time the head remains submerged!
 *
 * In seconds.
 */
#define TASK_MEAN_TIME 7

/** @brief The time the head will remain submerged before reading the sensors.
 *
 * In seconds.
 */
#define TASK_SAMPLE_TIME 5

/** @brief Initialise task module dependencies.
 */
void task_init();

/** @brief Set the automatic sampling rate.
 *
 * @param[in,out] t The new settings; on error, the maximum allowable value of
 * each member is set.
 * @returns @c 0, if the settings were acceptable (and, thus, stored); @c -1,
 * otherwise, in which case @p task is altered to contain the maximum values.
 */
int8_t task_set(Task* t);

/** @brief Get the current configuration for automated sampling.
 */

```

```
*  
* @param[out] t See corresponding parameter of task_set().  
*/  
void task_get(Task* t);  
  
/**  
* @brief Initiate a chain of samplings.  
*  
* The positions to be sampled are determined by the internal algorithm.  
*  
* @param[in] count The total amount of samples to take.  
*/  
void task_log_samples(uint8_t count);  
  
/**  
* @brief Initiate a single sampling at the specified position.  
*  
* Note that this may fail if axis Z is manipulated externally.  
*  
* @param[in] pos The position to sample. Member @c z is ignored.  
* @returns @c 0, if @p pos is valid; @c -1, otherwise.  
*/  
uint8_t task_log_sample(Position* pos);  
  
/**  
* @brief Returns whether there are registered tasks still in progress.  
*  
* @returns @c 0, if there are not; non-zero, otherwise.  
*/  
uint8_t task_pending();  
  
/**  
* @brief Return the estimate time for the completion of pending tasks.  
*  
* @returns The estimate in seconds. @c 0 denotes no estimate or no pending  
* tasks (to disambiguate, use task_pending()).  
*/  
uint16_t task_get_estimate();  
  
/**  
* @brief Create an acceptable random coordinate.  
*  
* The coordinates returned by this function respect the current operating range  
* (see, motor_get_max()). The initial values of @p x and @p y are used as part  
* of the seeding.  
*  
* @param[in, out] x The generated @c x position.  
* @param[in, out] y The generated @c y position.
```

```

*/
static void make_target(uint8_t* x, uint8_t* y);

/**
 * @brief Calculate the time it should take to get the motors to @p new position.
 *
 * @param[in] new The intended motor position.
 * @returns The estimated time in seconds. This includes any pending tasks to
 * @c 0, if no estimate can be given (in case the motors are moving or
 * already there); @c 0xFF
 */
static uint16_t task_estimate_time(Position* new);

/**
 * @brief Motor event handler.
 *
 * It is responsible for performing the requested amount of measurements and
 * logging their results.
 *
 * @param[in] pos The position of the device head, as given by #motor_callback.
 * @param[in] evt Status code describing the nature of the event, as given by
 * #motor_callback.
 */
static void task_handle_motor(Position pos, uint8_t evt);

#endif /* TASK_H_INCL */

task.c

#include "task.h"
#include "defs.h"
#include "log.h"
#include "util.h"
#include "motor.h"
#include "sensor.h"

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

/**
 * @brief Convert BCD hours @p h and minutes @p m to an interval since midnight.
 *
 * Note that each interval unit equals 6 minutes.
 */
#define BCD8_TO_INTERVAL(h, m) (FROM_BCD8(h) * (60/6) + FROM_BCD8(m) / 6)

/**
 * @brief Pending samples.

```

```
*  
* This is set by task_log_sample() and task_log_samples() and managed by  
* task_handle_motor().  
*/  
static uint8_t pending_samples;  
  
/**  
* @brief Return value of task_pending().  
*/  
static uint8_t task_is_pending;  
  
/**  
* @brief Time-stamp of the most recent measurement.  
*  
* This value is calculated at start-up (when task_init() is invoked) and updated  
* after each new log record is appended (in task_handle_motor()).  
*  
* Each unit equals 6 minutes. Values range in @c 0---@c 240 (just like  
* Task#interval).  
*/  
static uint8_t task_recent;  
  
/**  
* @brief Time-stamp of the current task operation.  
*  
* Note that this refers to the *seconds* elapsed since the beginning of this (or  
* the previous) hour at which time #task_estimate was set.  
*/  
static uint16_t task_start;  
  
/**  
* @brief The estimated time to complete the current task (in seconds).  
*/  
static uint16_t task_estimate;  
  
/**  
* @brief Current settings of automated samplings.  
*/  
static Task task;  
  
void task_init() {  
    /* Default date limits. */  
    BCDDate since = {.year = 0x00, .mon = 0x01, .date = 0x01,  
                    .hour = 0x00, .min = 0x00, .sec = 0x00};  
    BCDDate until = {.year = 0x99, .mon = 0x12, .date = 0x31,  
                     .hour = 0x23, .min = 0x59, .sec = 0x59};  
    LogRecordSet set;  
    LogRecord rec;
```

```

        uint8_t count;

        /* Identify the time-stamp of the most recent sampling. */
        count = log_get_set(&set, &since, &until);
        if(count) {
            /* Fetch the last record. */
            log_get_next(&rec, &set);

            task_recent = BCD8_TO_INTERVAL(rec.date.hour, rec.date.min);

            DBG_printf("Recent:@%02x:%02x-%d\n",
                      rec.date.hour,
                      rec.date.min,
                      task_recent));

        } else {
            task_recent = 0;
        }

        motor_set_callback(&task_handle_motor);
    }

    int8_t task_set(Task* t) {
        if(t->interval > 240) {
            t->interval = 240;
            t->samples = 255;
            return -1;
        }

        task.interval = t->interval;
        task.samples = t->samples;
        return 0;
    }

    void task_get(Task* t) {
        t->interval = task.interval;
        t->samples = task.samples;
    }

    void task_log_samples(uint8_t count) {
        if(count) {
            Position pos = {0, 0, 0};

            /* Request the first random position and begin sampling. */
            make_target(&pos.x, &pos.y);

            pending_samples = count;
            task_is_pending = 1;
        }
    }
}

```

```
        motor_set(pos);
    }
}

uint8_t task_log_sample(Position* pos) {

    /* Request the sensor head be submerged. */
    pos->z = 0;

    /* Schedule a single sampling. It is important this be done before calling
     * motor_set() because, this way, a correct estimate can be calculated by
     * update_motor_eta(). */
    pending_samples = 1;

    /* Stop, if the position is not valid. */
    if(motor_set(*pos)) {
        pending_samples = 0;
        return -1;
    }

    task_is_pending = 1;

    return 0;
}

uint8_t task_pending() {
    return task_is_pending;
}

uint16_t task_get_estimate() {
    BCDDate now;
    uint8_t day;
    uint16_t now_stamp;

    if(!task_estimate) return 0;

    get_date(&now, &day);
    now_stamp = FROM_BCD8(now.min) * 60 + FROM_BCD8(now.sec);

    /* If the task was initiated in the previous hour, add an hour (in seconds)
     * to @c now_stamp. Then, subtract then to get the interval. */
    if(now_stamp < task_start) now_stamp += 3600;

    now_stamp -= task_start;

    /* Subtract the interval from the estimate. */
    if(now_stamp > task_estimate) return 0;
    return task_estimate - now_stamp;
}
```

```

}

static void make_target(uint8_t* x, uint8_t* y) {
    BCDDate dt;
    uint8_t day;
    Position max;
    uint8_t seed;

    get_date(&dt, &day);
    motor_get_max(&max);
    seed = dt.sec + dt.min + *x + *y;
    *x = seed % max.x;
    *y = (seed + day) % max.y;
}

static uint16_t task_estimate_time(Position* new) {
    uint16_t time;
    Position cur;
    int16_t one;
    int16_t two;

    if(!motor_get(&cur)) {

/* printf("[%d, %d, %d] →[%d, %d, %d]\n", cur.x, cur.y, cur.z,*/
/* new->x, new->y, new->z);*/

        /* Preserve the offset on either X or Y, whichever is greater (since the
         * common part of the two is run in parallel). */
        one = abs(cur.x - new->x);
        two = abs(cur.y - new->y);
        if(one < two) one = two;

        /* Add a hypothetical delay to reach each target that is generated by
         * make_target(). */
        if(pending_samples > 1) one += TASK_MEAN_TIME * pending_samples;

        /* Increment overall offset by two times the dimension of Z (ie, to go
         * down and back up) plus the time the head remains submerged for as many
         * times as there are pending tasks. Note that it takes 1s to translate
         * by 1. */
        two = (GRID_Z_LEN * 2 + TASK_SAMPLE_TIME) * pending_samples;

        /* If @c z is at @c 0, then the head is already lowered and the sampling
         * has been performed; the latter being certain because this function is
         * called once it is needed to *raise* the head away from @c 0 (and only
         * then is cur.z equal to @c 0). That last sample is, also, already
         * removed from @c pending_samples. Still, the time it takes to fully
         * retract head for that sampling should be taken into account. */
    }
}

```

```
if(!cur.z) two += GRID_Z_LEN;

/* printf(''ETA: (%d+%d): %d\n'', one, two, one+two);*/
    time = one + two;
} else {
    time = 0;
}
return time;
}

static void task_handle_motor(Position pos, uint8_t evt) {

switch(evt) {
case MTR_EVT_BUSY: {
    BCDDate now;
    uint8_t day;
    get_date(&now, &day);

    task_is_pending = 1;
    task_estimate = task_estimate_time(&pos);
    task_start = FROM_BCD8(now.min) * 60 + FROM_BCD8(now.sec);

} break;
case MTR_EVT_OK:
    if(pending_samples) {
        /* Take a sample, if the sensor head is submerged. */
        if(pos.z == 0) {
            LogRecord rec;
            Position max;
            uint8_t day;
            uint16_t t;

            _delay_ms(TASK_SAMPLE_TIME * 1000);
            t = sens_read_t();

            get_date(&rec.date, &day);
            /* Prepare record. */
            rec.t = t >> 3;
            rec.x = pos.x;
            rec.y = pos.y;
            rec.rh = 0xFF;
            rec.ph = 0xFF;

            /* Log the result. */
            log_append(&rec);

            /* Update #task_recent now because a reset may be issued
             * before any of the remaining tasks are completed. */
        }
    }
}
```

```

task_recent = BCD8_TO_INTERVAL(rec.date.hour,
                                rec.date.min);

/* Since the measurement is complete, the head should be
 * retracted. */
motor_get_max(&max);
pos.z = max.z - 1;

/* Check whether there are pending samples to take and
 * request a pair of X-Y coordinates. Note that retracting
 * the sensor head (ie, increasing translation along axis Z)
 * takes precedence over motion along axes X and Y. Thus, it
 * is not necessary to set Z and X-Y motor coordinates
 * separately (see motor_update()). */
if(--pending_samples) {

    /* Request a new random X-Y position for the head. */
    make_target(&pos.x, &pos.y);
}

/* The head has reached a new position and there are pending
 * samples. Request the head be submerged to take a sample later
 * on. */
} else {
    pos.z = 0;
}

motor_set(pos);
} else {
    task_is_pending = 0;
    task_estimate = 0;
}
break;
}

/**
 * @brief Periodic automated sampling ISR.
 *
 * This is the Watchdog Timer ISR, responsible for waking the CPU every 8s (the
 * maximum interval for this MCU) to check whether there are tasks that should be
 * initiated automatically. They are only initiated granted the following:
 * — No other task is currently in progress.
 * — Task interval and samples have been specified (each, other than @c 0). See
 * task_set().
 * — The RTC is running.
 * — The elapsed quanta since the most recently performed task are equal or
 * greater the the specified interval (as set with task_set()).

```

```

*/
ISR(WDT_vect) {
    BCDDate now;
    uint8_t day;
    uint16_t now_stamp;
    uint16_t elapsed; /* Allow for a two-day interval. */

    DBG_printf("pending: %d, interval: %d, samples: %d\n", task_is_pending, task.interval, task.samples));
    _delay_ms(100);

    /* Do not proceed, if a task is in progress or there are no automation
     * settings. */
    if(task_is_pending || !task.interval || !task.samples) return;

    /* Also, do not proceed if the (RTC) clock is not running. */
    get_date(&now, &day);

    if(bit_is_set(now.sec, RTC_CH)) return;

    /* Calculate the interval between the most recent sampling and the current
     * time-stamp. */
    now_stamp = BCD8_TO_INTERVAL(now.hour, now.min);

    /* This is for when #task_recent is before midnight and @c now_stamp is
     * after. */
    if(now_stamp < task_recent) now_stamp += 240;

    elapsed = now_stamp - task_recent;

    DBG_printf("Now (%02x:%02x): %d\n"
              "Most recent: %d\n"
              "Elapsed: %d\n", now.hour, now.min, now_stamp, task_recent, elapsed);

    if(elapsed >= task.interval) {
        task_log_samples(task.samples);

        /* printf("Samples : %d\n", task.samples); */
    }

    /* puts("'); */
    _delay_ms(100);
}

```

## Ορισμοί TWI (I<sup>2</sup>C)

### twi.h

```

/**
 * @file

```

```

*/
#ifndef TWI_H_INCL
#define TWI_H_INCL

#include "defs.h"

/** 
 * @brief Busy-waits until @c TWINT of @c TWCR register is set. Status code is
 * read with #TWI_STATUS().
 */
#define TWI_WAIT() while(!(TWCR & _BV(TWINT)));

/** 
 * @brief Clears @c TWINT and sets forth the next action.
 */
#define TWI_DO() TWCR = _BV(TWINT) | _BV(TWEN);

/** 
 * @brief Clears @c TWINT and sets forth the next action, enabling transmission
 * of @c ACK upon completion.
 *
 * This should be preferred over #TWI_DO() when, in Master Receiver mode, it is
 * required to acknowledge the received byte.
 */
#define TWI_DO_ACK() TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN);

/** 
 * @brief A #TWI_DO() followed by an #TWI_WAIT().
 */
#define TWI_DO_WAIT() TWI_DO();\
                    TWI_WAIT()

/** 
 * @brief Gains possession of the bus.
 *
 * Busy-waits until the operation of performed. #TWI_STATUS() should used to
 * identify the outcome.
 */
#define TWI_START() TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN);\
                    TWI_WAIT()

/** 
 * @brief Releases the bus.
 *
 * It sets the appropriate bits and instantly returns.
 */
#define TWI_STOP() TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN)

```

```
/**  
 * @brief Reads a byte from the bus in @p x.  
 *  
 * The LSB of @p x is set to indicate a @c READ operation.  
 *  
 * @param[in] x Data byte read from the bus.  
 */  
#define TWI_SLA_R(x) TWDR = x | 1;\  
    TWI_DO_WAIT()  
  
/** @brief Sends data contained in @p x over the TWI bus.  
 *  
 * The LSB of @p x is cleared to indicate a @c WRITE operation.  
 *  
 * @param[in] x Data byte to send over the bus.  
 */  
#define TWI_SLA_W(x) TWDR = x & 0xFE;\  
    TWI_DO_WAIT()  
  
/**  
 * @brief Return the TWI bus status code.  
 *  
 * The last two bits are the prescaler bits; the bit before them is reserved and  
 * always reads zero (*Atmel p236*). Regardless, all three (last) bits are masked  
 * when reading the status code.  
 */  
#define TWI_STATUS() (TWSR & 0xF8)  
  
/**  
 * @brief TWI bus status code; @c START condition has been transmitted.  
 * *Atmel p222,p225*  
 *  
 * In Master Transmitter (MT) mode, other possible status codes include:  
 * #TWI_RSTART, #TWI_ARB_LOST, #TWI_SLA_W_ACK, #TWI_SLA_W_NACK, #TWI_DATA_W_ACK  
 * and #TWI_DATA_W_NACK.  
 *  
 * In Master Receiver (MR) mode, other possible status codes include:  
 * #TWI_RSTART, #TWI_ARB_LOST, #TWI_SLA_R_ACK, #TWI_SLA_R_NACK, #TWI_DATA_R_ACK  
 * and #TWI_DATA_R_NACK.  
 */  
#define TWI_SSTART (0x08)  
  
/**  
 * @brief TWI bus status code; repeated START condition has been transmitted.  
 * *Atmel p222,p225*  
 *  
 * Used both in Master Transmitter (MT) and Master Receiver (MR) mode. For a full
```

```
* list of codes per mode, see #TWI_SSTART.  
*/  
#define TWI_RSTART (0x10)  
  
/**  
* @brief TWI bus status code; arbitration lost in SLA+R mode or NOT ACK bit  
* (MR); arbitration lost in SLA+W or data bytes (MT). *Atmel p222,p225*  
*  
* Used both in Master Transmitter (MT) and Master Receiver (MR) mode. For a full  
* list of codes per mode, see #TWI_SSTART.  
*/  
#define TWI_ARB_LOST (0x38)  
  
/**  
* @brief TWI bus status code; SLA+W has been transmitted and ACK has  
* been received. *Atmel p222*  
*  
* Other status codes in Master Transmitter (MT) mode: #TWI_SSTART, #TWI_RSTART,  
* #TWI_ARB_LOST, #TWI_SLA_W_NACK, #TWI_DATA_W_ACK and #TWI_DATA_W_NACK.  
*/  
#define TWI_SLA_W_ACK (0x18)  
  
/**  
* @brief TWI bus status code; SLA+W has been transmitted and NOT ACK has  
* been received. *Atmel p222*  
*  
* Other status codes in Master Transmitter (MT) mode: #TWI_SSTART, #TWI_RSTART,  
* #TWI_ARB_LOST, #TWI_SLA_W_ACK, #TWI_DATA_W_ACK and #TWI_DATA_W_NACK.  
*/  
#define TWI_SLA_W_NACK (0x20)  
  
/**  
* @brief TWI bus status code; data byte has been transmitted, ACK has been  
* received. *Atmel p222*  
*  
* Other status codes in Master Transmitter (MT) mode: #TWI_SSTART, #TWI_RSTART,  
* #TWI_ARB_LOST, #TWI_SLA_W_ACK, #TWI_SLA_W_NACK and #TWI_DATA_W_NACK.  
*/  
#define TWI_DATA_W_ACK (0x28)  
  
/**  
* @brief TWI bus status code; data byte has been transmitted, NOT ACK has been  
* received. *Atmel p222*  
*  
* Other status codes in Master Transmitter (MT) mode: #TWI_SSTART, #TWI_RSTART,  
* #TWI_ARB_LOST, #TWI_SLA_W_ACK, #TWI_SLA_W_NACK and #TWI_DATA_W_ACK.  
*/  
#define TWI_DATA_W_NACK (0x30)
```

```
/**  
 * @brief TWI bus status code; SLA+R has been transmitted, ACK has been received.  
 * @Atmel p225*  
  
 * Other status codes in Master Receiver (MR) mode: #TWI_SSTART, #TWI_RSTART,  
 * #TWI_ARB_LOST, #TWI_SLA_R_NACK, #TWI_DATA_R_ACK and #TWI_DATA_R_NACK.  
 */  
#define TWI_SLA_R_ACK (0x40)  
  
/**  
 * @brief TWI bus status code; SLA+R has been transmitted, NOT ACK has been  
 * received. *Atmel p225*  
  
 * Other status codes in Master Receiver (MR) mode: #TWI_SSTART, #TWI_RSTART,  
 * #TWI_ARB_LOST, #TWI_SLA_R_ACK, #TWI_DATA_R_ACK and #TWI_DATA_R_NACK.  
 */  
#define TWI_SLA_R_NACK (0x48)  
  
/**  
 * @brief TWI bus status code; data byte has been received, ACK has been  
 * returned. *Atmel p225*  
  
 * Other status codes in Master Receiver (MR) mode: #TWI_SSTART, #TWI_RSTART,  
 * #TWI_ARB_LOST, #TWI_SLA_R_ACK, #TWI_SLA_R_NACK and #TWI_DATA_R_NACK.  
 */  
#define TWI_DATA_R_ACK (0x50)  
  
/**  
 * @brief TWI bus status code; data byte has been received, NOT ACK has been  
 * returned. *Atmel p225*  
  
 * Other status codes in Master Receiver (MR) mode: #TWI_SSTART, #TWI_RSTART,  
 * #TWI_ARB_LOST, #TWI_SLA_R_ACK, #TWI_SLA_R_NACK and #TWI_DATA_R_ACK.  
 */  
#define TWI_DATA_R_NACK (0x58)  
  
/**  
 * @brief Executes @c x, returning @c -1 upon failure.  
 *  
 * Upon executing @p x, #TWI_STATUS() is used to determine the outcome of the  
 * operation. Should the status code returned by it be unequal to @c s, the bus  
 * is released (#TWI_STOP()) and @c -1 is returned. Note that this is a macro  
 * and, so, the returning function is the 'callee'.  
 *  
 * @param[in] x Arbitrary code to execute. Typically, one of the TWI convenience  
 * macros.  
 * @param[in] s Status code that is expected upon successful completion of @p x.
```

```

*/
#define TWI_ATTEMPT(x, s) x; \
    if(TWI_STATUS() != s) {\ \
        TWI_STOP(); \
        return -1; \
    }

/** 
 * @brief Initializes the TWI registers.
 *
 * This macro disables TWI and clears the @c STOP condition bit. It then sets the
 * TWI bit-rate value and the prescaler bits (see #TWBR_VALUE and #TWI_RATE).
 * The reason behind first disabling TWI is to make sure the
 * MCU may operate upon it even after entering Power-down mode (as noted in
 * *Atmel p.210*, all devices connected to the TWI bus must be powered to allow
 * any bus operation).
 */
#define TWI_INIT() TWCR &= ~(_BV(TWSTO) | _BV(TWEN)); \
    TWBR = TWBR_VALUE; \
    TWSR |= (TWI_RATE & (_BV(TWPS1) \
    | _BV(TWPS0)));
#endif /* TWI_H_INCL */

```

## Ολοκληρωμένο δικτύωσης W5100

### w5100.h

```

/** 
 * @file
 * @brief API for the network module W5100.
 * @addtogroup network_w5100 Network module (W5100)
 * @{
 *
 * @brief API for the network module W5100.
 */
#ifndef W5100_H_INCL
#define W5100_H_INCL

#include <inttypes.h>

/** 
 * @brief Mode register.
 *
 * Responsible for software reset and enabling ping block, PPPoE, Indirect bus
 * auto-increment and Indirect bus mode.
 *
 * See bits: #NET_MR_RST, #NET_MR_PB.

```

```
/*
#define NET_MR 0x0000

/***
 * @brief Reset bit; if set, the module will reset.
 *
 * Automatically cleared, once reset has completed.
 */
#define NET_MR_RST 7

/***
 * @brief Ping Block mode bit; if set, ping responses are enabled.
 */
#define NET_MR_PB 4

/* There are more bits specified which are irrelevant to the implementation. */

/***
 * @brief Default Gateway Address Register.
 *
 * 4-Bytes long.
 */
#define NET_GAR 0x0001

/***
 * @brief Subnet Mask Register.
 *
 * 4-Bytes long.
 */
#define NET_SUBR 0x0005

/***
 * @brief Source Hardware Address Register.
 *
 * 6-Bytes long.
 */
#define NET_SHAR 0x0009

/***
 * @brief Source IP Address Register.
 *
 * 4-Bytes long.
 */
#define NET_SIPR 0x000F

/***
 * @brief Interrupt Register (read-only).
 *

```

```

* Contains interrupt flags. Pin @c nINT remains low for as long as there is at
* least one such bit set, granted that its mask bit is also set (see #NET_IMR).
*
* To set which events cause an interrupt, use #NET_IMR.
*/
#define NET_IR 0x0015

/**
* @brief Interrupt Mask Register.
*
* Specifies under which circumstances an interrupt is generated. Each bit in
* this register corresponds to the interrupt flag bit of #NET_IR at the same
* position.
*
* See bits: #NET_IR_S0
*/
#define NET_IMR 0x0016

/**
* @brief Interrupt coming from Socket n.
*
* @param[in] n Socket to produce interrupt (@c 0---@c 3).
*/
#define NET_IR_Sn(n) (1<<n)

/* There are more bits specified which are irrelevant to the implementation. */

/**
* @brief Rx Memory Size Register; default 0x55.
*
* Distributes the available buffer for incoming data to the specified sockets.
* The total memory is 8KB. Size of Socket 0 is determined by bits @c 0 and @c 1,
* and so on.
*
* See #NET_SIZEn().
*/
#define NET_RMSR 0x001A

/**
* @brief Tx Memory Size Register; default 0x55.
*
* Distributes the available buffer for outgoing data to the specified sockets.
* The total memory is 8KB. Size of Socket 0 is determined by bits @c 0 and @c 1,
* and so on.
*
* See #NET_SIZEn().
*/
#define NET_TMSR 0x001B

```

```
/**  
 * @brief #NET_TMSR and #NET_RMSR combination of 1KB.  
 *  
 * To create the appropriate value for a specific Socket, use #NRW_SIZEn().  
 */  
#define NET_SIZE_1 0x00  
  
/**  
 * @brief #NET_TMSR and #NET_RMSR combination of 2KB.  
 *  
 * To create the appropriate value for a specific Socket, use #NRW_SIZEn().  
 */  
#define NET_SIZE_2 0x01  
  
/**  
 * @brief #NET_TMSR and #NET_RMSR combination of 4KB.  
 *  
 * To create the appropriate value for a specific Socket, use #NRW_SIZEn().  
 */  
#define NET_SIZE_4 0x02  
  
/**  
 * @brief #NET_TMSR and #NET_RMSR combination of 8KB.  
 *  
 * To create the appropriate value for a specific Socket, use #NRW_SIZEn().  
 */  
#define NET_SIZE_8 0x03  
  
/**  
 * @brief Convenience macro to create Socket size bits.  
 *  
 * @param[in] n Socket number (@c 0---@c 3).  
 * @param[in] x Socket size (one of #NET_SIZE_1, #NET_SIZE_2, #NET_SIZE_4,  
 * #NET_SIZE_8).  
 */  
#define NET_SIZEn(n, x) (x << (n*2))  
  
/**  
 * @brief The start address of the Tx buffer.  
 */  
#define NET_TX_BASE 0x4000  
  
/**  
 * @brief The start address of the Rx buffer.  
 */  
#define NET_RX_BASE 0x6000
```

```

/** 
 * @brief Base address offset of a socket.
 *
 * Each Socket is controlled by a set amount of registers. They span for @c 0x100
 * (@c 256) bytes starting at a *base* address. For instance, Socket 0 starts at
 * @c 0x400, Socket 1 starts at @c 0x500, and so on. This helps calculate the
 * base address for the specified Socket.
 *
 * @param[in] n Socket to calculate its base address (@c 0---@c 3).
 */
#define NET_Sn_OFFSET(n) (n*0x100 + 0x400)

/** 
 * @brief Socket Mode Register.
 *
 * Controls the operation of this Socket.
 *
 * See: #NET_Sn_MR_CLOSED, #NET_Sn_MR_TCP.
 */
#define NET_Sn_MR(n) (0x00 + NET_Sn_OFFSET(n))

/** 
 * @brief Disable this Socket.
 *
 * Used with #NET_Sn_MR().
 */
#define NET_Sn_MR_CLOSED (0x0)

/** 
 * @brief Use Socket in TCP mode.
 *
 * Used with #NET_Sn_MR().
 */
#define NET_Sn_MR_TCP (0x1)

/** 
 * @brief Socket Command Register.
 *
 * Accepts operation command to perform.
 *
 * See: #NET_Sn_CR_OPEN, #NET_Sn_CR_LISTEN, #NET_Sn_CR_CONNECT,
 * #NET_Sn_CR_DISCON, #NET_Sn_CR_CLOSE, #NET_Sn_CR_SEND, #NET_Sn_CR_RECV.
 */
#define NET_Sn_CR(n) (0x01 + NET_Sn_OFFSET(n))

/** 
 * @brief Initialise Socket.
 *

```

```
* Used with NET_Sn_CR().  
*/  
#define NET_Sn_CR_OPEN 0x01  
  
/**  
 * @brief Listen for incoming connection requests (TCP mode).  
 *  
 * Used with NET_Sn_CR().  
 */  
#define NET_Sn_CR_LISTEN 0x02  
  
/**  
 * @brief Connect to remote host (TCP mode).  
 *  
 * Used with NET_Sn_CR().  
 */  
#define NET_Sn_CR_CONNECT 0x03  
  
/**  
 * @brief Send connection termination request (TCP mode).  
 *  
 * Used with NET_Sn_CR().  
 */  
#define NET_Sn_CR_DISCON 0x08  
  
/**  
 * @brief Close Socket (change value of #NET_Sn_SR() to #NET_SN_SR_CLOSED).  
 *  
 * Used with NET_Sn_CR().  
 */  
#define NET_Sn_CR_CLOSE 0x10  
  
/**  
 * @brief Notify to send data from the outgoing buffer of the Socket.  
 *  
 * Used with NET_Sn_CR().  
 */  
#define NET_Sn_CR_SEND 0x20  
  
/**  
 * @brief Notify that data have been read from the incoming buffer of the Socket.  
 *  
 * Used with NET_Sn_CR().  
 */  
#define NET_Sn_CR_RECV 0x40  
  
/**  
 * @brief Interrupt flags of this Socket.  
 */
```

```
*  
* Each bit specifies a particular condition. See: #NET_Sn_IR_SEND_OK,  
* #NET_Sn_IR_TIMEOUT, #NET_Sn_IR_RECV, #NET_Sn_IR_DISCON, #NET_Sn_IR_CON.  
*/  
#define NET_Sn_IR(n) (0x02 + NET_Sn_OFFSET(n))  
  
/**  
* @brief Transmission is completed.  
*  
* Used with #NET_Sn_IR().  
*/  
#define NET_Sn_IR_SEND_OK 4  
  
/**  
* @brief Time-out (of connection or data transmission).  
*  
* Used with #NET_Sn_IR().  
*/  
#define NET_Sn_IR_TIMEOUT 3  
  
/**  
* @brief Available data.  
*  
* This bit remains set for as long as there data available for this Socket (even  
* after executing #NET_Sn_CR_RECV).  
*  
* Used with #NET_Sn_IR().  
*/  
#define NET_Sn_IR_RECV 2  
  
/**  
* @brief Connection termination is requested or completed.  
*  
* Used with #NET_Sn_IR().  
*/  
#define NET_Sn_IR_DISCON 1  
  
/**  
* @brief Connection established.  
*  
* Used with #NET_Sn_IR().  
*/  
#define NET_Sn_IR_CON 0  
  
/**  
* @brief Status flags of this Socket.  
*  
* Each bit specifies a particular state. See: #NET_Sn_SR_CLOSED,
```

```
* #NET_Sn_SR_INIT, #NET_Sn_SR_LISTEN, #NET_Sn_SR_ESTAB, #NET_Sn_SR_CLOSEWAIT.  
*/  
#define NET_Sn_SR(n) (0x03 + NET_Sn_OFFSET(n))  
  
/**  
 * @brief Connection is terminated.  
 *  
 * Used with NET_Sn_SR(). See *w5100 p.29*.  
 */  
#define NET_Sn_SR_CLOSED 0x00  
  
/**  
 * @brief This occurs after #NET_Sn_CR_OPEN is given.  
 *  
 * Used with NET_Sn_SR(). See *w5100 p.30*.  
 */  
#define NET_Sn_SR_INIT 0x13  
  
/**  
 * @brief Socket is listening for incoming connections.  
 *  
 * Used with NET_Sn_SR(). See *w5100 p.30*.  
 */  
#define NET_Sn_SR_LISTEN 0x14  
  
/**  
 * @brief Connection established; data may now be received and sent.  
 *  
 * Used with NET_Sn_SR(). See *w5100 p.30*.  
 */  
#define NET_Sn_SR_ESTAB 0x17  
  
/**  
 * @brief Termination request has been received.  
 *  
 * Used with NET_Sn_SR(). See *w5100 p.30*.  
 */  
#define NET_Sn_SR_CLOSEWAIT 0x1C  
  
/**  
 * @brief Port number of this Socket.  
 *  
 * 2-Bytes long.  
 *  
 * Each bit specifies a particular state. See: #NET_Sn_SR_CLOSED,  
 * #NET_Sn_SR_INIT, #NET_Sn_SR_LISTEN, #NET_Sn_SR_ESTAB, #NET_Sn_SR_CLOSEWAIT.  
 */  
#define NET_Sn_PORT(n) (0x04 + NET_Sn_OFFSET(n))
```

```
/**  
 * @brief Socket TX Free Size Register.  
 *  
 * 2-Bytes long.  
 */  
#define NET_Sn_TX_FSR(n) (0x20 + NET_Sn_OFFSET(n))  
  
/**  
 * @brief Socket TX Read Pointer Register (read-only).  
 *  
 * 2-Bytes long.  
 */  
#define NET_Sn_TX_RR(n) (0x22 + NET_Sn_OFFSET(n))  
  
/**  
 * @brief Socket TX Write Pointer Register.  
 *  
 * 2-Bytes long.  
 */  
#define NET_Sn_TX_WR(n) (0x24 + NET_Sn_OFFSET(n))  
  
/**  
 * @brief Socket RX Received Size Register (read-only).  
 *  
 * 2-Bytes long.  
 */  
#define NET_Sn_RX_RSR(n) (0x26 + NET_Sn_OFFSET(n))  
  
/**  
 * @brief Socket RX Read Pointer Register.  
 *  
 * 2-Bytes long.  
 */  
#define NET_Sn_RX_RR(n) (0x28 + NET_Sn_OFFSET(n))  
  
/**  
 * @brief Initialise Socket buffer sizes.  
 *  
 * This should be called at least once before attempting to operate the Sockets.  
 * Setting #NET_RMSR and #NET_TMSR separately is not required. Calling this  
 * function should be preferred, instead, because it also initialises some  
 * internal settings.  
 *  
 * Care should be when allocating more than 2KB to a Socket; the total available  
 * memory for either Tx and Rx buffers is 8KB; *shared by all* sockets.  
 *  
 * Also, see #NET_SIZEn().
```

```
*  
* @param[in] Value of #NET_TMSR.  
* @param[in] Value of #NET_RMSR.  
*/  
void net_socket_init(uint8_t tx, uint8_t rx);  
  
/**  
* @brief Initialises a Socket.  
*  
* @param[in] s The Socket to initialise (@c 0--@c 3).  
* @param[in] mode The mode of operation of @p s. See #NET_Sn_MR().  
* @param[in] port The port number Socket @p s should operate on.  
*/  
void net_socket_open(uint8_t s, uint8_t mode, uint16_t port);  
  
/**  
* @brief Prepare the SPI bus to communicate with the W5100.  
*  
* This function:  
* — Disables SPI (in case it was running).  
* — Sets up the appropriate clock rate (see #NET_SPSR and #NET_SPCR).  
* — Enables the chip (see #NET_ENABLE()).  
* — Delays 1us for @c nCS setup time (*W5100 p.67*).  
* — Does *not* enable the SPI clock!  
*/  
void net_select();  
  
/**  
* @brief Convenience function to send a byte over to the W5100.  
*  
* @param[in] addr The address to write to.  
* @param[in] data The data to send.  
*/  
void inline net_write8(uint16_t addr, uint8_t data);  
  
/**  
* @brief Convenience function to read a byte over from the W5100.  
*  
* @param[in] addr The address to read from.  
* @returns The data read.  
*/  
uint8_t net_read8(uint16_t addr);  
  
/**  
* @brief Convenience function to read a word over from the W5100.  
*  
* @param[in] addr The address to start reading from.  
* @returns The data to read.  
*/
```

```
/*
uint16_t net_read16(uint16_t addr);

/** 
 * @brief Convenience function to send a word over to the W5100.
 *
 * @param[in] addr The address to start writing to.
 * @param[in] data The data to send.
 */
void net_write16(uint16_t addr, uint16_t data);

/** 
 * @brief Wrapper around net_exchange() to send data to the W5100.
 *
 * It is safe to assume that, upon completion, the contents of @p buf will not
 * have been altered.
 *
 * Also, see net_exchange().
 *
 * @param[in] addr The address to start writing to. It is incremented for each
 * byte sent.
 * @param[in] buf The data to send.
 * @param[in] len The amount of bytes to write.
 */
void inline net_write(uint16_t addr, uint8_t* buf, uint16_t len);

/** 
 * @brief Wrapper around net_exchange() to read data from the W5100.
 *
 * Also, see net_exchange().
 *
 * @param[in] addr The address to start reading from. It is incremented for each
 * byte received.
 * @param[out] buf The data read.
 * @param[in] len The amount of bytes to read.
 */
void inline net_read(uint16_t addr, uint8_t* buf, uint16_t len);

/** 
 * @brief Exchange the specified amount of bytes starting at @p addr.
 *
 * It is safe to assume that, if a write command is specified (@c 0xF0), the
 * contents of @p buf will not have been altered.
 *
 * @param[in] c @c 0xF0 to @c write to, @c 0x0F to @c read data from the W5100.
 * @param[in] addr The address to start writing/reading to/from. It is
 * incremented for each byte sent/received.
 * @param[in,out] buf The data to send. Upon completion, it contains the data
 */
```

```

* received, if @p c was @c 0x0F.
* @param[in] len The amount of bytes to write/read.
*/
void net_exchange(uint8_t c, uint16_t addr, uint8_t* buf, uint16_t len);

/**
* @brief Send data to a W5100 Socket output buffer.
*
* It is safe to call this function with a @p len of @c 0 and @p flush of
* non-zero to sent any previously set W5100 buffer data on socket @p s. In this
* case, @p buf could be @c NULL.
*
* @param[in] s The socket to send data to.
* @param[in] buf Array containing bytes to send. This should be at least @p len
* bytes long.
* @param[in] The number of bytes to copy from @p buf into the W5100 output
* buffer.
* @param[in] flush Designates whether data in @buf along with all previously
* unsent data of socket @p s should be sent out with this call.
* @returns @c The available space in the output buffer of W5100 for socket @c s
* (after appending @p len bytes from @p buf), if @p flush was @c 0; the socket
* size (in bytes), if the buffer was just flushed (ie, @p flush was non-zero);
* a negative number for the amount of bytes that cannot fit into the available
* space.
*/
uint16_t net_send(uint8_t s, uint8_t* buf, uint16_t len, uint8_t flush);

/**
* @brief Receive data from a W5100 Socket input buffer.
*
* @param[in] s The socket to read data from.
* @param[in] buf Array of bytes read. This should be at least @p len bytes long.
* @param[in] The number of bytes to read from the W5100 buffer.
* @returns @c The available bytes in the input buffer (after reading @p len
* bytes).
*/
uint16_t net_recv(uint8_t s, uint8_t* buf, uint16_t len);

#endif /* W5100_H_INCL */
/** @}

```

**w5100.c**

```

#include "w5100.h"
#include "defs.h"

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

```

```
/**  
 * @brief The absolute Tx address for each socket.  
 *  
 * Initialised by net_socket_init().  
 */  
static uint16_t tx_base[4];  
  
/**  
 * @brief The address Tx mask for each socket.  
 *  
 * This value helps identify the relative offset within each buffer. Initialised  
 * by net_socket_init().  
 */  
static uint16_t tx_mask[4];  
  
/**  
 * @brief The absolute Rx address for each socket.  
 *  
 * Initialised by net_socket_init().  
 */  
static uint16_t rx_base[4];  
  
/**  
 * @brief The address Rx mask for each socket.  
 *  
 * This value helps identify the relative offset within each buffer. Initialised  
 * by net_socket_init().  
 */  
static uint16_t rx_mask[4];  
  
/**  
 * @brief The amount of data stored in W5100 output buffer for each socket.  
 *  
 * Used by net_send() to facilitate sending data to W5100 without necessarily  
 * flushing them.  
 */  
static uint16_t socket_contents[4];  
  
void net_socket_init(uint8_t tx, uint8_t rx) {  
    uint16_t tx_sum = 0; /* Sum of previously allocated Tx buffers. */  
    uint16_t rx_sum = 0; /* Sum of previously allocated Rx buffers. */  
    uint16_t size; /* Size of current buffer (either Tx or Rx). */  
    uint8_t i;  
  
    net_write(NET_TMSR, &tx, 1);  
    net_write(NET_RMSR, &rx, 1);
```

```
/* Calculate the absolute start address of each Socket (sub)buffer as well
 * as the mask (Tx and Rx). */
for(i = 0 ; i < 4 ; ++i) {

    /* The start address of this Socket's (sub)buffers depends on the size
     * of all previously allocated (sub)buffers (Tx and Rx). */

    size = 1024 << (tx & 0x03);
    tx_base[i] = NET_TX_BASE + tx_sum;
    tx_mask[i] = size - 1;
    tx_sum += size;

    size = 1024 << (rx & 0x03);
    rx_base[i] = NET_RX_BASE + rx_sum;
    rx_mask[i] = size - 1;
    rx_sum += size;

    /* Prepare the size bits for the next Socket. */
    tx >>= 2;
    rx >>= 2;
}

void net_socket_open(uint8_t s, uint8_t mode, uint16_t port) {

    /* Close Socket @p s. */
    net_write8(NET_Sn_CR(s), NET_Sn_CR_CLOSE);
    net_write8(NET_Sn_MR(s), mode);
    net_write16(NET_Sn_PORT(s), port);
    net_write8(NET_Sn_CR(s), NET_Sn_CR_OPEN);

    /* Wait for the Socket to be opened. #NET_Sn_CR() clears automatically once
     * the command is executed. *W5100 p.27* */
    do {
        } while(net_read8(NET_Sn_CR(s)));
}

void net_select() {
    /* Disable SPI, if running. */
    SPCR = 0;

    /* Use the specified clock settings in SPI Master mode (0,0). Do *not*
     * enable. */
    SPSR = NET_SPSR & _BV(SPI2X);
    SPCR = (NET_SPCR & (_BV(SPR1) | _BV(SPRO))) | _BV(MSTR);
}

void inline net_write8(uint16_t addr, uint8_t data) {
```

```

    net_exchange(0xF0, addr, &data, 1);
}

uint8_t net_read8(uint16_t addr) {
    uint8_t data;
    net_exchange(0x0F, addr, &data, 1);
    return data;
}

uint16_t net_read16(uint16_t addr) {
    uint16_t data;
    data = ((uint16_t)net_read8(addr)) << 8;
    data |= net_read8(addr + 1);
    return data;
}

void net_write16(uint16_t addr, uint16_t data) {
    net_write8(addr, data >> 8);
    net_write8(addr + 1, data);
}

void inline net_write(uint16_t addr, uint8_t* buf, uint16_t len) {
    net_exchange(0xF0, addr, buf, len);
}

void inline net_read(uint16_t addr, uint8_t* buf, uint16_t len) {
    net_exchange(0x0F, addr, buf, len);
}

void net_exchange(uint8_t c, uint16_t addr, uint8_t* buf, uint16_t len) {
    uint8_t update = c == 0x0F;
    uint8_t byte;
    uint16_t i;

    net_select();
    SPCR |= _BV(SPE);

    for(i = 0 ; i < len ; ++i) {
        /* Set the appropriate signals to select the W5100. Wait for a minimum
         * of 21ns before sending any CLK pulses. *W5100 p.67* */
        NET_ENABLE();

        SPDR = c;
        loop_until_bit_is_set(SPSR, SPIF);

        SPDR = addr>>8;
        loop_until_bit_is_set(SPSR, SPIF);
    }
}

```

```
SPDR = addr & 0x00FF;
loop_until_bit_is_set(SPSR, SPIF);

SPDR = buf[i];
loop_until_bit_is_set(SPSR, SPIF);

if(update) {
    buf[i] = SPDR;
}

++addr;

NET_DISABLE();
}

/* Delay before releasing control. *W5100 p.67* */
_delay_us(1);
SPCR &= ~_BV(SPE);
}

uint16_t net_send(uint8_t s, uint8_t* buf, uint16_t len, uint8_t flush) {
    uint16_t free_size = net_read16(NET_Sn_TX_FSR(s));
    uint16_t s_content = socket_contents[s];

    /* Is there enough available space? */
    if(free_size < s_content + len) return free_size - s_content - len;

    /* Send data from local buffer to W5100 buffer. */

    uint16_t tx_WR = net_read16(NET_Sn_TX_WR(s));

    /* Offset from the (sub)buffer base. */
    uint16_t tx_offset = (tx_WR + s_content) & tx_mask[s];

    /* Physical address to start writing to. */
    uint16_t start_addr = tx_base[s] + tx_offset;
    uint16_t sock_size = tx_mask[s] + 1;

    /* If the requested bytes (len) exceed the buffer limit, then overflow will
     * occur. Write data from current offset up to limit (upper-bound), and
     * then, the remainder of bytes starting from the (sub)buffer base. */
    if((tx_offset + len) > sock_size) {
        /* Bytes until upper-bound. */
        uint16_t bound = sock_size - tx_offset;

        /* Write bytes up to upper-bound. */
        net_write(start_addr, buf, bound);
    }
}
```

```

/* Write the rest of the bytes starting off from the base. */
net_write(tx_base[s], buf + bound, len - bound);

} else {
    net_write(start_addr, buf, len);
}

/* Update WR pointer for future operations. */

s_content += len;
if(flush) {
    uint8_t status;
    tx_WR += s_content;

    net_write16(NET_Sn_TX_WR(s), tx_WR);
    net_write8(NET_Sn_CR(s), NET_Sn_CR_SEND);

    do {
        status = net_read8(NET_Sn_IR(s));
    } while((status & NET_Sn_IR_SEND_OK) != NET_Sn_IR_SEND_OK);

    s_content = 0;
}
socket_contents[s] = s_content;
return sock_size - s_content;
}

uint16_t net_recv(uint8_t s, uint8_t* buf, uint16_t len) {
    uint16_t rx_size = net_read16(NET_Sn_RX_RSR(s));
    uint16_t rx_RR = net_read16(NET_Sn_RX_RR(s));

    /* Read at most @p len bytes. */
    if(len < rx_size) rx_size = len;

    /* Offset from (sub)buffer base. */
    uint16_t rx_offset = rx_RR & rx_mask[s];

    /* Physical address to start reading from. */
    uint16_t start_addr = rx_base[s] + rx_offset;
    uint16_t sock_size = rx_mask[s] + 1;

    /* If incoming (data size + current read offset) exceeds buffer limit, then
     * overflow has occurred. Read data from the current offset up to limit, and
     * then, the remainder of bytes from (sub)buffer base. */
    if(rx_offset + rx_size > sock_size) {
        /* Bytes until upper-bound. */
        uint16_t bound = sock_size - rx_offset;

```

```

/* Read bytes up to upper-bound. */
net_read(start_addr, buf, bound);

/* Read the rest of the bytes, starting off from the base. */
net_read(rx_base[s], buf + bound, rx_size - bound);
} else {
    net_read(start_addr, buf, rx_size);
}

/* Update RR pointer for future reads. */
net_write16(NET_Sn_RX_RR(s), rx_RR + rx_size);
net_write8(NET_Sn_CR(s), NET_Sn_CR_RECV);

return rx_size - len;
}

```

## Συναρτήσεις γενικού σκοπού

### util.h

```

/**
 * @file
 * @brief General utility functions.
 */

#ifndef UTIL_H_INCL
#define UTIL_H_INCL

#include "rtc.h"
#include "defs.h"

#include <inttypes.h>

#ifndef NULL
/** 
 * Specify that a pointer has not been set to a valid address.
 */
#define NULL 0
#endif

/** 
 * @brief Convert tens @p t and units @p u into BCD notation.
 *
 * @param[in] t Tens. Ranges in @c 0---@c 9.
 * @param[in] u Units. Ranges in @c 0---@c 9.
 */
#define TO_BCD8(t, u) ((t << 4) | u)

*/

```

```

* @brief Convert BCD @p hex into a decimal.
*
* @param[in] hex A number in BCD notation. It should contain no digits above @c
* 9.
*/
#define FROM_BCD8(hex) ((hex >> 4)*10 + (hex & 0x0F))

/**
* @brief Convert an unsigned integer to string.
*
* Note that @p buf is the *last* address to write to; the actual array should
* expand before it.
*
* @param[out] buf The array into which to store the result. The address provided
* is the *last* address that will be used in the conversion (and which will
* contain @c \0. The array should be large enough to contain the result
* (inclusive of null-byte).
* @param[in] number The number to convert into string.
* @returns The amount of digits written (null-byte not included).
*/
uint8_t uint_to_str(uint8_t* buf, uint16_t number);

/**
* @brief Convert a temperature reading into a string.
*
* @p t is expected to contain the integral part of the number in the 7 most
* significant bits. Bit @c 0 is the fraction @f$ 2^{-1} @f$. For more
* information, see #LogRecord.
*
* @param[out] buf The result of the conversion (null-terminated).
* @param[in] len Size of @p buf. Although specified, @p buf should be large
* enough to accommodate the result, regardless.
* @param[in] t The value to convert.
* @returns The amount of bytes written (non-inclusive of null-byte).
*/
uint8_t temp_to_str(uint8_t* buf, uint8_t len, uint8_t t);

/**
* @brief Convert and IP address string into four bytes.
*
* Uses <stdlib.h>strtol().
*
* @param[out] ip An array of four bytes to write to.
* @param[in] buf An IP address string (null-terminated).
* @returns @c 0, if four, dot-separated, numbers have been parsed; non-zero,
* otherwise.
*/
uint8_t str_to_inet(uint8_t* ip, uint8_t* buf);

```

```
/**  
 * @brief Convert an array of integers into a string, each separated with a dot.  
 *  
 * It receives an array of four bytes and converts them to an equivalent IP  
 * address string (null-terminated).  
 *  
 * More specifically, for each address byte, an initial position is estimated and  
 * then incremented depending on the number of its digits. Then, a character is  
 * passed into @p buf for each of its digits (using modulo and quotient).  
 *  
 * @param[out] buf The string array to write to.  
 * @param[in] ip A four-byte array of an IP address (or mask).  
 * @returns The number of bytes written into @p buf (non-inclusive of null-byte).  
 */  
uint8_t inet_to_str(uint8_t* buf, uint8_t* ip);  
  
/**  
 * @brief Read the current date and time from the RTC.  
 *  
 * @param[out] dt The current date and time.  
 * @param[out] day Day of week; @c 1 denotes Sunday.  
 */  
void get_date(BCDDate* dt, uint8_t* day);  
  
/**  
 * @brief Set the current date and time of the RTC.  
 *  
 * @param[in] dt The current date and time.  
 * @param[in] day Day of week; @c 1 denotes Sunday.  
 */  
void set_date(BCDDate* dt, uint8_t day);  
  
/**  
 * @brief Read string into an #RTCMap variable.  
 *  
 * Currently, the string is parsed up to seconds (not including fraction).  
 * Note that this function does not check the validity of the date as a whole  
 * (eg, days of month), but rather, that each value does not exceed a maximum  
 * allowed value.  
 *  
 * @param[out] dt  
 * @param[in] buf String in ISO8601 format (YYYY-MM-DDTHH:mm:ss.sssZ).  
 * @returns @c 0, if parsing the date was successful; @c 0, otherwise.  
 */  
int8_t str_to_date(BCDDate* dt, uint8_t* buf);
```

```

/** 
 * @brief Convert the supplied date and time into an ISO8601-formatted string.
 *
 * Although <stdio.h>printf() could be used to format the date, it is chosen not
 * to, because this way, it results in smaller code footprint, no need to use the
 * stack (for the variable arguments) or store the format string.
 *
 * @param[out] buf An array, at least 25 bytes wide, that accepts the ISO8601
 * format of the supplied date and time (YYYY-MM-DDTHH:mm:ss.sssZ). The string
 * will be null-terminated. Fraction of a second always reads zero; time-zone
 * is set to @c Z (UTC).
 * @param[in] dt Date to convert into string.
 */
void date_to_str(uint8_t* buf, BCDDate* dt);

/** 
 * @brief Read a number of program memory chunks into @p buf.
 *
 * Accepts a variable amount of addresses of strings that reside in program
 * memory. The strings are loaded into @p buf and the address of the first byte
 * of each is placed into @p indices.
 *
 * Some notes:
 * — The last argument *should* always be @c NULL.
 * — @p indices should be at least as large as the number of string addresses
 * given (excluding @c NULL).
 * — @p buf should be large enough to accommodate all requested strings.
 *
 * @param[out] indices Array of strings read from program memory.
 * @param[out] buf Buffer into which to store the strings read from program
 * memory.
 * @param[in] ... Program memory addresses to read strings from. The last one
 * *should* be @c NULL.
 * @returns The amount of bytes written. In other words, the starting offset at
 * which further strings may be written, if required.
 */
uint16_t pgm_read_str_array(uint8_t** indices, uint8_t* buf, ...);

/** 
 * @brief Map an #RTCMap to a #BCDDate variable.
 *
 * It is used in-line by get_date().
 *
 * @param[out] dt Conversion destination.
 * @param[in] rtc Source.
 */
static inline void rtc_to_date(BCDDate* dt, RTCMap* rtc);

```

```

/** 
 * @brief Map a #BCDDate to an #RTCMMap variable.
 *
 * It is used in-line by set_date().
 * Note that @link RTCMap#date date@endlink will not be set.
 *
 * @param[out] rtc Conversion destination.
 * @param[in] dt Source.
 */
static inline void date_to_rtc(RTCMap* rtc, BCDDate* dt);

/** 
 * @brief Transfer data from Flash to the network module.
 *
 * The network module should have enough available buffer space to accommodate
 * the outgoing data. This function does not flush the output buffer.
 *
 * @param[in] Socket of the network module to write to.
 * @param[in] page The page to start reading from.
 * @param[in] len The number of bytes to send.
 */
void fls_to_wiz(uint8_t s, uint16_t page, uint16_t len);

#endif /* UTIL_H_INCL */

```

**util.c**

```

#include "util.h"
#include "rtc.h"
#include "flash.h"

#include <avr/pgmspace.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>

uint8_t uint_to_str(uint8_t* buf, uint16_t number) {
    uint8_t unit;
    uint8_t i = 0;

    *buf = '\0';
    while(number > 0) {
        ++i;
        *(buf + i) = number % 10 + '0';
        number /= 10;
    }
    if(i == 0) {
        *(buf + 1) = '0';
        i = 1;
    }
}

```

```

    }

    return i;
}

uint8_t temp_to_str(uint8_t* buf, uint8_t len, uint8_t t) {
    uint8_t digits;
    uint8_t i;

    digits = uint_to_str(buf + len - 1, t >> 1);
    for(i = 0 ; i < digits ; ++i) {
        buf[i] = buf[len - 1 - digits + i];
    }
    buf[i++] = '.';
    buf[i++] = t & 0x01 ? '5' : '0';
    buf[i] = 0;

    return i;
}

uint8_t str_to_inet(uint8_t* ip, uint8_t* buf) {
    uint8_t i;
    uint8_t* j = buf;
    char* last = NULL;

    for(i = 0 ; i < 4 && !last ; ++i) {
        ip[i] = strtol(j, &last, 10);

        if(*last == '.') {
            j = last + 1;
            last = NULL;
        }
    }
    if(i == 4) {
        return 0;
    }
    return 1;
}

uint8_t inet_to_str(uint8_t* buf, uint8_t* ip) {
    uint8_t byte; /* A single byte from @p ip. */
    uint8_t i; /* For each byte in @p ip. */
    uint8_t pos; /* Position in @p buf to write to next. */
    uint8_t j; /* Digit of @c byte to write next. */

    for(i = 0, pos = 0 ; i < 4 ; ++i) {
        byte = ip[i];
        j = 0;
        if(byte >= 10 && byte < 100) ++pos;
    }
}

```

```
else if(byte >= 100) pos += 2;

/* Ensure this runs at least once so that a single zero may not be
 * omitted. */
do {
    buf[pos - j] = byte % 10 + '0';
    ++j;
    byte /= 10;
} while(byte);

/* Increment for next '.' or terminating null-byte. */
++pos;
/* If more bytes are to follow, place a '.' and further increase @c pos
 * to point at the position to write the next digit to. */
if(i != 3) buf[pos++] = '.';

buf[pos] = '\0';
return pos;
}

void get_date(BCDDate* dt, uint8_t* day) {
    RTCMap rtc;
    rtc_get(&rtc);
    rtc_to_date(dt, &rtc);
    *day = rtc.day;
}

void set_date(BCDDate* dt, uint8_t day) {
    RTCMap rtc;
    date_to_rtc(&rtc, dt);
    rtc.day = day;
    rtc_set(&rtc);
}

int8_t str_to_date(BCDDate* dt, uint8_t* buf) {
    uint8_t num = 0;
    uint8_t error = 0;

    /* @p buf should contain a string of at least a full-date without fractions
     * of a second. */
    if(strlen(buf) < 19) return 1;

    error += buf[0] != '2';
    error += buf[1] != '0';

    error += !isdigit(buf[2]);
    error += !isdigit(buf[3]);
    dt->year = TO_BCD8(buf[2] - '0', buf[3] - '0');
```

```

error += buf[4] != '-';
error += !isdigit(buf[5]);
error += !isdigit(buf[6]);
dt->mon = TO_BCD8(buf[5] - '0', buf[6] - '0');
error += dt->mon > 0x12;

error += buf[7] != '-';
error += !isdigit(buf[8]);
error += !isdigit(buf[9]);
dt->date = TO_BCD8(buf[8] - '0', buf[9] - '0');
error += dt->date > 0x31;

error += buf[10] != 'T';
error += !isdigit(buf[11]);
error += !isdigit(buf[12]);
dt->hour = TO_BCD8(buf[11] - '0', buf[12] - '0');
error += dt->hour > 0x23;

error += buf[13] != ':';
error += !isdigit(buf[14]);
error += !isdigit(buf[15]);
dt->min = TO_BCD8(buf[14] - '0', buf[15] - '0');
error += dt->min > 0x59;

error += buf[16] != ':';
error += !isdigit(buf[17]);
error += !isdigit(buf[18]);
dt->sec = TO_BCD8(buf[17] - '0', buf[18] - '0');
error += dt->sec > 0x59;

return error > 0;
}

void date_to_str(uint8_t* buf, BCDDate* dt) {
    /* This is equivalent to the following: @verbatim
    sprintf(buf, "20%02x-%02x-%02xT%02x:%02x:%02x.000Z", dt->year,
            dt->mon,
            dt->date,
            dt->hour,
            dt->min,
            dt->sec);@endverbatim
    * but is faster, requires less memory and does not force a dependency on
    * vprintf(). */
    buf[0] = '2';
    buf[1] = '0';
    buf[2] = '0' + ((dt->year & 0xF0) >> 4);
    buf[3] = '0' + (dt->year & 0x0F);
}

```

```

buf[4] = '-';
buf[5] = '0' + ((dt->mon & 0xF0) >> 4);
buf[6] = '0' + (dt->mon & 0x0F);
buf[7] = '-';
buf[8] = '0' + ((dt->date & 0xF0) >> 4);
buf[9] = '0' + (dt->date & 0x0F);
buf[10] = 'T';
buf[11] = '0' + ((dt->hour & 0xF0) >> 4);
buf[12] = '0' + (dt->hour & 0x0F);
buf[13] = ':';
buf[14] = '0' + ((dt->min & 0xF0) >> 4);
buf[15] = '0' + (dt->min & 0x0F);
buf[16] = ':';
buf[17] = '0' + ((dt->sec & 0xF0) >> 4);
buf[18] = '0' + (dt->sec & 0x0F);
buf[19] = '.';
buf[20] = '0';
buf[21] = '0';
buf[22] = '0';
buf[23] = 'Z';
buf[24] = 0;
}

uint16_t pgm_read_str_array(uint8_t** indices, uint8_t* buf, ...) {
    va_list ap; /* Pointer to each optional argument. */
    PGM_P str; /* Address of a string in Flash. */
    uint8_t i = 0; /* Index of @p indices to write to. */
    uint8_t* init = buf;

    va_start(ap, buf);
    str = va_arg(ap, PGM_P);

    /* Repeat for all supplied progmem strings. */
    while(str != NULL) {
        indices[i] = buf; /* Store start address of the string. */

        strcpy_P(buf, str); /* Copy string into temporary buffer. */
        buf += strlen(buf) + 1; /* Prepare the next buffer address. +1
        * is for null-byte. */

        ++i;
        str = va_arg(ap, PGM_P);
    }
    return buf - init;
}

static inline void rtc_to_date(BCDDate* dt, RTCMap* rtc) {
    dt->year = rtc->year;
}

```

```

dt->mon = rtc->mon;
dt->date = rtc->date;
dt->hour = rtc->hour;
dt->min = rtc->min;
dt->sec = rtc->sec;
}

static inline void date_to_rtc(RTCMap* rtc, BCDDate* dt) {
    rtc->year = dt->year;
    rtc->mon = dt->mon;
    rtc->date = dt->date;
    rtc->hour = dt->hour;
    rtc->min = dt->min;
    rtc->sec = dt->sec;
}

void fls_to_wiz(uint8_t s, uint16_t page, uint16_t len) {
    uint16_t size = 256;
    uint8_t buf[256];

    while(len) {
        if(len < size) size = len;

        /* Read @c size bytes from the Flash. */
        fls_exchange(FLS_READ, page, buf, size);

        /* Send them to the W5100 HTTP server output buffer. */
        net_send(s, buf, size, 0);

        /* Prepare for the next iteration. */
        ++page;
        len -= size;
    }
}

```

## B.2 Διεπαφή χρήστη («ιστοσελίδα»)

### Αρχικοποίηση JavaScript client

ui/client.js

```

(function(ns) {

    /* Initialise Logger. */
    ns.Logger.init({ "idLog" : "infobox",
                    "entries" : 10,
                    "expires" : 0.5,
                    "clsEntry" : ""});

```

```
    "clsNew" : "new",
    "clsInfo" : "info",
    "clsCritical" : "critical",
    "clsFatal" : "fatal"
});

/* Request and display device date in Logger. */
var req = ns.createRequest();
if(!req) return;

req.open("GET", "configuration.php");
req.onreadystatechange = function() {
    var conf,
        date,
        day,
        el;

    if(this.readyState !== 4) return;

    conf = JSON.parse(this.responseText);

    date = new Date(conf.date);
    date = ns.fixInt(date.getUTCDate(), 2) + "-"
        + ns.fixInt(date.getUTCMonth(), 2) + "-"
        + ns.fixInt(date.getUTCFullYear(), 2) + ","
        + date.getUTCHours() + ":"
        + ns.fixInt(date.getUTCMinutes(), 2) + ":"
        + ns.fixInt(date.getUTCSeconds(), 2);

    day = conf.day;

    days = ['Κυρ', 'Δευ', 'Τρι', 'Τετ', 'Πεμ', 'Παρ', 'Σαβ'];

    el = document.getElementById("device-time");
    el.innerHTML = days[day - 1] + "," + date;

    /* Also, reveal the element's parent which is, by default, hidden (so as
     * not to be displayed empty, in case Javascript is disabled). */
    el.parentNode.className = ns.replaceWord(el.parentNode.className,
        "hidden",
        '');
};

req.send();

/**
 * @brief Update the document's title for PageHome.
 */

```

```

function pageStaticHome() {
    window.document.title = 'ΤΑΙΟΛΙΚΝΟ-Γενικά';
}

/**
 * @brief Update the document's title for PageHelp.
 */
function pageStaticHelp() {
    window.document.title = 'ΤΑΙΟΛΙΚΝΟ-Βοήθεια';
}

/* The area of search results (in page Log). Contains @c .total-count and
 * @c .pagination elements. */
var lrs = document.getElementById('log-section-result');

/* Uses for @c pages, @c menu and @c PageMonitor. */
var pageNames = ['home', 'log', 'config', 'operate', 'help'];

/* Handlers for each page. @c null is for simple-text (static) pages. */
var handlers = [pageStaticHome,
                ns.PageLog.reload,
                ns.PageConfig.reload,
                ns.PageOperate.reload,
                pageStaticHelp];

var pages = new ns.OneOfMany(pageNames,
                             document.getElementById('content')
                             .querySelectorAll('.page'),
                             'page-visible',
                             'page-hidden',
                             'config');

var menu = new ns.OneOfMany(pageNames,
                           document.getElementById('nav')
                           .querySelectorAll('li'),
                           'nav-menu-current',
                           'nav-menu',
                           'config');

/* Initialise Pages. */

ns.PageLog.init({ 'hash' : '#log',
                  'form' : {
                      'idSize' : 'log-page-size',
                      'idSince' : 'log-since',
                      'idUntil' : 'log-until',
                      'clsError' : 'field-msg-error'
                  },
                  'data' : {

```

```
        "elTotal" : lrs.querySelector('.total-count'),
        "elPages" : lrs.querySelector('.pagination'),
        "clsCurrent" : 'button-current',
        "clsOther" : 'button',
        "idSection" : "log-section-result",
        "clsHidden" : "hidden",
        "idTable" : "log-table"
    });
}

ns.PageConfig.init({ 'form' : {
    "idIAddr" : "config-iaddr",
    "idGateway" : "config-gateway",
    "idSubnet" : "config-subnet",
    "idDate" : "config-date",
    "idConfigX" : "config-x",
    "idConfigY" : "config-y",
    "idConfigZ" : "config-z",
    "idIntervalHrs" : "config-interval-hours",
    "idIntervalMins" : "config-interval-minutes",
    "idSamples" : "config-samples",
    "clsError" : "field-msg-error"
}});

ns.PageOperate.init({ 'form' : {
    "idNewX" : "operate-x",
    "idNewY" : "operate-y",
    "clsError" : "field-msg-error"
},
"data" : {
    "idStatus" : "operate-status",
    "idRange" : "operate-range"
}});

/* Initialise PageMonitor (the swapper). */
ns.PageMonitor.init(pageNames, handlers, pages, menu, 'home');

/* Register event callbacks. */
ns.addEventListener(document.getElementById('log-submit'),
    'click',
    ns.PageLog.submit);

ns.addEventListener(document.getElementById('log-reset'),
    'click',
    ns.PageLog.reset);

ns.addEventListener(document.getElementById('config-save'),
    'click',
    ns.PageConfig.submit);
```

```

ns.addEventListener(document.getElementById("config-reset"),
                  "click",
                  ns.PageConfig.reset);

ns.addEventListener(document.getElementById("config-reload"),
                  "click",
                  ns.PageConfig.reload);

ns.addEventListener(document.getElementById("operate-move"),
                  "click",
                  ns.PageOperate.move);

ns.addEventListener(document.getElementById("operate-sample"),
                  "click",
                  ns.PageOperate.sample);

/* Add listeners to update the page content when the URI fragment is
 * altered. Reloading the state 'onhashchange' responds to hash changes due
 * to history traversal (eg, pressing back or forward), whereas 'onload'
 * deals with direct visiting (eg, a bookmarked link or page refresh). */
ns.addEventListener(window, "hashchange", ns.PageMonitor.update);
ns.addEventListener(window, "load", ns.PageMonitor.update);

})(window.gNS);

```

## **Κοινόχρηστος κώδικας client**

### **ui/gns-common.js**

```

;(function(ns) {

    /**
     * @brief Allows to select one among a collection of peers.
     *
     * Calling switchTo() with a key found in @p arrKeys, results in applying
     * class @p clsActive to the element @p arrEls in the corresponding index.
     * Any previously selected element is automatically applied @p clsInactive.
     * At most one element may be selected at a time.
     *
     * @param[in] arrKeys Strings to identify each element in @arrEls (on a
     * one-on-one basis).
     * @param[in] arrEls Elements to be applied classes @p clsActive and @p
     * clsInactive.
     * @param[in] clsActive
     * @param[in] clsInactive
     * @param[in] keyDefault Key of the element to select upon instantiation.
     * Optional.
     */

```

```

ns.OneOfMany =
function (arrKeys, arrEls, clsActive, clsInactive, keyDefault) {
    this.one = null;
    this.many = {};
    this.active = clsActive;
    this.inactive = clsInactive;

    var i, // Index of @p arrKeys—arrEls pair
        el; // An element from @p arrEls

    for(i = 0 ; i < arrKeys.length ; ++i) {
        el = arrEls[i];
        el.className = clsInactive;
        this.many[arrKeys[i]] = el;
    }

    if(keyDefault !== undefined) {
        this.switchTo(keyDefault);
    }
};

ns.OneOfMany.prototype = {

    /**
     * @brief Selects the element that is paired with @p key.
     *
     * The current element (if any) is set to this.inactive regardless of
     * whether @p key corresponds to an available key.
     *
     * @param[in] key The key for the element to select. If @c null, none
     * will be selected.
     */
    switchTo : function(key) {
        if(this.one !== null) {
            this.one.className = this.inactive;
            this.one = null;
        }

        if(this.many[key]) {
            this.one = this.many[key];
            this.one.className = this.active;
        }
    }
};

/**
 * @brief Facilitates managing pagination controls.
 *
 * Every instance keeps track of an element that displays the number results

```

```
* and a second element for page anchors (pagination). Upon invoking show(),
* the 'innerHTML' of those elements is updated accordingly.
*
* The link ('href') of each page anchor is generated by an external
* function,@p fnLinker. @p clsCurrent is a class string to apply to the page
* anchor dubbed as current (or selected), while @p clsOther is applied to
* every other.
*
* @param[in] elTotal Element which innerHTML will be set to the amount of
* search results.
* @param[in] elPages Element which innerHTML will be populated with a number
* of anchors.
* @param[in] clsCurrent The currently displayed anchor page will be applied
* this class.
* @param[in] clsOther All anchors, except for the selected one, will be
* applied this class.
* @param[in] fnLinker A function that accepts a page number/index (starting
* at @c 1) and returns an appropriate value for its corresponding anchor's
* 'href' attribute.
*/
ns.Paginator =
function (elTotal, elPages, clsCurrent, clsOther, fnLinker) {
    this.elTotal = elTotal;
    this.elPages = elPages;
    this.clsCurrent = clsCurrent;
    this.clsOther = clsOther;
    this.linker = fnLinker;
};
ns.Paginator.prototype = {

    /**
     * @brief Update the pagination display.
     *
     * @param[in] results The amount of total results.
     * @param[in] anchors The amount of page links to generate.
     * @param[in] current The index of the currently displayed page (starts
     * at @c 1). This page will not have its 'href' set, whereas its class
     * will be set to clsCurrent unlike the other anchors that acquire
     * clsOther.
     */
    show : function(results, anchors, current) {
        var i,
            limit,
            pages = "";

        this.elTotal.innerHTML = results;
        this.elPages.innerHTML = "";
```

```

i = 1;
limit = current;

while(i <= anchors) {
    for(i ; i < limit ; ++i) {
        pages += "<a class=''" + this.clsOther + "'>" +
            "href=''" + this.linker(i) + "'>" +
            i + "</a>";
    }

    if(i === current) {
        /* Set the class of the current anchor but do not provide
         * a link. */
        pages += "<a class=''" + this.clsCurrent + "'>" +
            + i + "</a>";
        ++i;
        limit = anchors + 1;
    }
}

if(pages === '') {
    pages = "<a class=''" + this.clsCurrent + "'>0</a>";
}
this.elPages.innerHTML = pages;
},

/**
 * @brief Set totals @c 0 and page anchors to a single &ndash;.
 */
reset : function() {
    this.elTotal.value = 0;
    this.elPages.innerHTML = "&ndash;";
}
};

/** 
 * @brief Expand @p child's prototype with @p parent's
 *
 * Prototype members than exist on both @p child and @p parent are ignored,
 * by default, unless @p override evaluates to @c true.
 *
 * @param[in,out] child Contains the prototype to be expanded.
 * @param[in,out] parent Contains the prototype to be copied.
 * @param[in] override Replace common prototype members with those of @p
 * parent.
 */
ns.augment = ns.augment ||
function (child, parent, override) {

```

```

var i;

for(i in parent.prototype) {
    if(child.prototype[i] !== undefined && !override) continue;
    child.prototype[i] = parent.prototype[i];
}

/***
 * @brief Register a callback function for a particular element.
 *
 * Currently, a single callback is supported in fallback mode.
 *
 * @param[in] el The element to be attached a listener.
 * @param[in] event String of the event name (non-inclusive of "on").
 * @param[in] fn The callback function / event handler.
 */
ns.addEventListener = ns.addEventListerner ||

function (el, event, fn) {
    if(el.addEventListener) {
        el.addEventListener(event, fn);

        /* This should suffice for IE versions prior to 9. */
    } else if(el.attachEvent) {
        el.attachEvent("on" + event, fn);

    } else {
        el["on" + event] = fn;
    }
};

/***
 * @brief Return an object that can be used to make HTTP request objects.
 *
 * If instantiation of XMLHttpRequest is not possible, Msxml2.XMLHTTP and
 * Microsoft.XMLHTTP will be attempted in that order. If none of these
 * works, @c null is returned. Also, a message is displayed using Logger.
 *
 * @returns An XML HTTP object or @c null.
 */
ns.createRequest = ns.createRequest ||
function () {
    var request = null;

    if(window.XMLHttpRequest) {
        request = new XMLHttpRequest();
    } else if(window.ActiveXObject) {

```

```

try {
    request = new ActiveXObject('Msxml2.XMLHTTP');

} catch(e) {
    try {
        request = new ActiveXObject('Microsoft.XMLHTTP');
    } catch(e) {
    }
}

if(request === null) {
    ns.log('Προέκυψε σοβαρό σφάλμα! Λενόγταν δυνατή λόγη με ουργία',
        + ' αντικείμενου σύνδεσης (XMLHTTPRequest). Φορτώστε πάλι την',
        + ' σελίδα προσπαθήστε με πιο σύγχρονο πρόγραμμα.', 'fatal');
}

return request;
};

/***
 * @brief Check whether @p value is NaN.
 *
 * This deals with the issue of isNaN() returning @c false even for
 * non-number values.
 *
 * @param[in] value The value to check.
 * @returns @c true if @p value is NaN; @p false, otherwise.
 */
ns.isNaN = ns.isNaN ||
function (value) {
    if(typeof value !== 'number') return false;
    return isNaN(value);
};

/***
 * @brief Check whether @p collection is empty.
 *
 * @param[in] collection The collection to check.
 * @returns @c true if @p collection is empty or @c null; @p false,
 * otherwise.
 */
ns.isEmpty = ns.isEmpty ||
function (collection) {
    var i = 0,
        val;
}

```

```

        for(val in collection) {
            ++i;
            break;
        }
        return i === 0;
    };

    /**
     * @brief Convert a number into a string of fixed digits.
     *
     * @param[in] num The number to prefix with zeros.
     * @param[in] digits The number of total digits to produce.
     * @returns A string number with that many @p digits.
     */
    ns.fixInt = ns.fixInt ||
    function (num, digits) {
        var value = num.toString(10),
            times = digits - value.length,
            i;

        for(i = 0 ; i < times ; ++i) {
            value = "0" + value;
        }
        return value;
    };

    /**
     * @brief Convert @p seconds into a string of format MMSS.
     *
     * Note that minutes and seconds are separated by the prime symbol (&#8242;)
     * and the seconds are followed by the double prime symbol (&#8243;).
     *
     * @param[in] seconds The number of seconds to convert into string.
     * @returns @p seconds as a string with format: MMSS
     */
    ns.minsec = ns.minsec ||
    function (seconds) {
        return ns.fixInt(Math.floor(seconds / 60), 2) + "&#8242;" +
               ns.fixInt(seconds % 60, 2) + "&#8243;";
    };

    /**
     * @brief Replace a word within a string with another.
     *
     * The default behaviour is to replace only the first occurrence of @p
     * substring with @p substitute.
     *
     * @param[in] source The initial string.

```

```
* @param[in] substring The string to search for in @p source or,
* alternatively, a RegExp object.
* @param[in] substitute The string to replace @p substring with.
*
* @returns The string resulting from the substitution.
*/
ns.replaceWord = ns.replaceWord ||
function (source, substring, substitute, first) {
    /* Append spaces to search the word near boundaries, substitute and,
     * then, trim. */
    return ("_" + source + "_").replace(substring, substitute)
        .replace(/^\s*|\s*$/g, '');
};

/**
* @brief A Logger Entry.
*
* A Logger Entry creates an actual DOM element to contain a message and also
* keeps a reference to that, along with a time-stamp of its creation. The
* message, including the specified time-stamp (or Now, if none was
* specified), is enclosed within a @c div element which is applied the
* class @c cls. The time-stamp may be used to determine the age of the
* entry.
*
* @param[in] msg The message of the entry (text or HTML).
* @param[in] cls The class of the entry (enclosing @c div). Optional.
* @param[in] stamp Date object. The time to link this entry with. Defaults
* to Now. Optional.
* @param hideTime If supplied and it evaluates to true, the generated
* element will not contain the time.
*/
ns.LoggerEntry =
function (msg, cls, stamp, hideTime) {
    var time;

    this.elEntry = document.createElement('div');

    !stamp && (stamp = new Date());
    this.stamp = stamp;

    cls && (this.elEntry.className = cls);
    this.elEntry.innerHTML = msg;

    if(!hideTime) {
        time = ns.fixInt(stamp.getHours(), 2)
            + ":" + ns.fixInt(stamp.getMinutes(), 2)
            + ":" + ns.fixInt(stamp.getSeconds(), 2);
        this.elEntry.innerHTML = time + "-";
    }
}
```

```

        }

        this.elEntry.innerHTML += msg;

    };

    /**
     * @brief Manages a simple message board.
     *
     * The messages are added as entries within the specified DOM element and
     * contain, on top of the message passed to log(), the time of its
     * invocation.
     *
     * Each new message entry is appended on top of the others, whereas only a
     * maximum number of entries are preserved, the oldest ones being removed
     * (any pre-existent elements within the board, included).
     *
     * For each entry, a severity level may be specified to apply one of three
     * user-defined classes. An additional class is always applied to each new
     * entry, allowing CSS selectors to mark newly-added entries in a special
     * way. This class is removed on a succeeding call to log(), provided the
     * specified amount of seconds have elapsed.
     *
     * If log() is invoked without prior initialisation of Logger, the message
     * will be displayed with alert().
     */
    ns.Logger = ns.Logger || (function () {

        var elLog,
            entries = [],
            children,
            entriesMax,
            expires,
            clsNew,
            clsInfo,
            clsCritical,
            clsFatal;

        var isInit; // Whether Logger has been initialised. If not, log
                    // will fallback to alert().

        /**
         * @brief Initialise the Logger.
         *
         * @c elLog is parsed and its current children elements are counted as
         * entries (although, they won't be subjected to class removal on
         * successive calls to log()). During this process, all excess children
         * will be removed; at most, the *first* @c entries children are
         * preserved.
         */
    });

```

```
* All settings are required.  
*  
* @param[in] s Object with initialisation settings. It should contain  
* the following keys, each with a value as follows:@verbatim {  
* idLog id of the message log element (should accept  
* Flow Content)  
* entries Maximum number of entries to display  
* expires Each entry will retain clsNew for at least this  
* many seconds.  
* clsInfo Class to apply to all information messages  
* clsCritical Class to apply to all critical messages  
* clsFatal Class to apply to all fatal messages  
* clsNew Additional class to apply to new entries  
*}@endverbatim  
*/  
var init = function (s) {  
    elLog = document.getElementById(s.idLog);  
    entriesMax = s.entries;  
    clsNew = s.clsNew;  
    clsInfo = s.clsInfo;  
    clsCritical = s.clsCritical;  
    clsFatal = s.clsFatal;  
    expires = s.expires * 1000;  
    children = 0;  
  
    var child = elLog.firstChild,  
        next;  
  
    while(child) {  
        /* Get the next child before removing this one. */  
        next = child.nextSibling;  
  
        if(children === entriesMax) {  
            elLog.removeChild(child);  
  
        } else {  
            ++children;  
        }  
        child = next;  
    }  
  
    isInit = true;  
};  
  
/**  
* @brief Add a new message to the message board.  
*  
* Every new message entry is applied @c clsNew to it class name. This
```

```

* class is temporary and is removed with the first invocation of log()
* *after* @c expires seconds have passed. If provided, @p severity
* specifies a base class that will be added to the entry. This class is
* permanent and the actual value applied in each case is specified
* during the initialisation of Logger (see init()).
*
* @param[in] msg The message to display. Optional.
* @param[in] severity String specifying the class of the entry, as
* follows:
* - "info" applies @c clsInfo
* - "critical" applies @c clsCritical
* - "fatal" applies @c clsFatal
*/
var log = function (msg, severity) {
    var entry,
        now = new Date(),
        cls = '';
    if(severity === 'info') {
        cls = clsInfo;
    } else if(severity === 'critical') {
        cls = clsCritical;
    } else if(severity === 'fatal') {
        cls = clsFatal;
    }

    /* Fallback mode. */
    if(!isInit) {
        alert(msg);
        return;
    }

    entry = new ns.LoggerEntry(msg, clsNew + '「' + cls, now);
    elLog.insertBefore(entry.elEntry, elLog.firstChild);

    unmarkOld(now);
    entries.push(entry);

    /* Ensure there are not too many entries. */
    if(children === entriesMax) {
        entries.shift();
        elLog.removeChild(elLog.lastChild);
    } else {
        ++children;
    }
};

/**

```

```
* @brief Unmark older entries, if necessary'.
*/
var update = function () {
    unmarkOld();
};

/***
* @brief Un-mark old entries.
*
* All entries that are older than @c expires seconds will have @c clsNew
* removed from the class name of their element. Note that this only
* recognises elements add with log().
*
* @param[in] now Date object, typically, the current time. If not
* specified, a new object will be created internally. Optional.
*/
var unmarkOld = function (now) {
    var i,
        el;

    now = now ? now.getTime() : new Date().getTime();
    for(i = entries.length - 1 ; i >= 0 ; --i) {

        if(now - entries[i].stamp.getTime() > expires) {
            el = entries[i].elEntry;

            /* Stop, on the first non-new entry. */
            if(entries[i].elEntry.className
                .indexOf(clsNew) === -1) break;

            el.className = ns.replaceWord(el.className, clsNew, '');
        }
    }
};

/***
* @brief Remove all entries from the message board.
*
* This removes every element within @c elLog.
*/
var reset = function () {
    entries = [];
    elLog.innerHTML = '';
};

return {"init" : init,
        "log" : log,
        "update": update,
```

```
        "reset" : reset};
    }());
ns.log = ns.Logger.log;

/**
 * @brief Facilitate validation and access to input fields.
 *
 * A Field is linked to some input elements. It may then be used to validate
 * and return their value displaying an appropriate error message, if
 * necessary, empty them or validate candidate values without actually
 * setting them.
 *
 * This is not a complete definition and may not be instantiated. It should
 * be augmented to provide specific functionality. Any of the provided
 * members should be replaced to that end.
 */
ns.Field =
function () {
    throw "Error:@InstantiatingField.";
};

ns.Field.prototype = {

    /**
     * @brief The id of the underlying input field(s).
     *
     * This is used to gain a reference to the actual input field(s) as well
     * as an indicator of the error message (see _showErrors()).
     */
    id : null,

    /**
     * @brief A reference to the underlying element(s).
     *
     * Field is defined to support a single input element, though it may be
     * augmented to support multiple fields, as well.
     */
    el : null,

    /**
     * @brief Reference to the generated error element.
     */
    elMsg : null,

    /**
     * @brief Class string to apply to the element containing the error.
     *
     * See _showErrors().
     */
}
```

```
/*
clsError: null,

/** 
 * @brief Set the value of this Field.
 *
 * The value only updates the underlying input fields, if it passes the
 * constraints of validate().
 *
 * @param[in] value The value to set to the Field.
 * @returns The value returned by validate().
 */
set : function(value) {
    value = this.validate(value);
    if(value !== null) {
        this.el.value = value;
    }
    return value;
},

/** 
 * @brief Parse the current value of the Field.
 *
 * To ensure the field remains visually correct when the value is also
 * correct, its contents are updated after parsing.
 *
 * @param[in, out] error Object. Designates that, if validation fails,
 * an error message should be added to the DOM. This object is added
 * a new member named after this.id. Its value is an array of error
 * strings. The generated DOM element (containing the error message)
 * may be accessed via this.elMsg. Do note that the element is
 * automatically appended to the DOM (see, _showErrors()). Optional.
 * @returns A valid value parsed from the underlying input fields or @c
 * null.
 */
get : function(errors) {
    var value = this.set(this.el.value);

    if(value === null && typeof errors === 'object') {
        errors[this.id] = this._showErrors();
    }

    return value;
},

/** 
 * @brief Clear the value of the input field(s).
 *
```

```
* @returns this
*/
reset : function() {
    this.el.value = '';
    return this;
},

/**
 * @brief Validate the supplied value and return it or return @c null.
 *
 * This function accepts a value to test against the constraints of this
 * Field.
 *
 * @return The validated value, which may not be exactly the same as
 * @p value (for instance, trimmed of leading and trailing spaces).
 */
validate : null,

/**
 * @brief Remove the error message from the DOM.
 *
 * @returns this
*/
resetMsg : function() {
    if(this.elMsg) {
        this.elMsg.parentNode.removeChild(this.elMsg);
        delete this.elMsg;
    }
    return this;
},

/**
 * @brief Class string applied to the generated error elements.
 *
 * @returns this
*/
setErrorClass : function(clsError) {
    this.clsError = clsError;
    return this;
},

/**
 * @brief Construct an appropriate error message for this Field.
 *
 * @returns An array of error strings.
 */
_getErrors : null,
```

```
/*
 * @brief Creates and appends an error element to the DOM.
 *
 * @returns The generated element.
 */
_showErrors : function() {
    var ul,
        errors = this._getErrors();

    /* Set error message, if validation fails. */
    ul = document.createElement('ul');
    ul.innerHTML = '<li>' +
        errors.join('</li><li>') +
        '</li>';

    this.clsError && (ul.className = this.clsError);

    /* Append the message. */
    this.el.parentNode.appendChild(ul);
    this.resetMsg();
    this.elMsg = ul;
    return ul;
}

/**
 * @brief Track an integer Field.
 *
 * Since a FieldInt requires a single input field, most of the core
 * functionality remains intact. Only validate() and _getErrors() are
 * implemented.
 *
 * @param[in] idInt id of the input element.
 * @param[in] min The minimum allowable value. The validation and the
 * generated error message are affected by this value. Optional.
 * @param[in] max The maximum allowable value. The validation and the
 * generated error message are affected by this value. Optional.
 */
ns.FieldInt =
function(idInt, min, max) {
    this.id = idInt;
    this.el = document.getElementById(idInt);
    this.min = min;
    this.max = max;
};
ns.FieldInt.prototype = {

    /**

```

```

* @brief Parse @p value as an integer within this.min, this.max.
*
* If this.min or this.max are @c undefined, they do not affect the
* validation.
*
* @returns A valid integer (within this.min and this.max); @c null, on
* error.
*/
validate : function(value) {
    value = parseInt(value, 10);

    if(ns.isNaN(value)
    || (this.min !== undefined && this.min > value)
    || (this.max !== undefined && this.max < value))
        return null;
    return value;
},

/**
* @brief Constructs an error message for this Field's constraints.
*
* 'FieldInt' only returns a single string (wrapped in an array) adjusted
* to the value of this.min and this.max of an instance at the time of
* its invocation.
*
* @returns An array of string errors.
*/
_getErrors : function() {
    /* Construct the appropriate error message. */
    var msg = '';

    /* Construct the tail of the error *without* the period. */
    if(this.min !== undefined && this.max !== undefined) {
        msg = "\u0391\u039d\u039d\u039c\u039d" + this.min + "\u039c\u0395\u039e\u0391\u039d" + this.max;

    } else if(this.min !== undefined) {
        msg = "\u0391\u039d\u039d\u039c\u039d" + this.min + "\u039c\u0391\u039d\u0391\u039d\u039c\u0391\u039d";

    } else if(this.max !== undefined) {
        msg = "\u039c\u0395\u039e\u0391\u039d" + this.max;
    }

    /* Combine the constant part, the tail (if any) and the period.*/
    msg = "Απαιτείται \u0391\u039d\u039d\u039c\u039d" + msg + ".";
    return [msg];
}

};

ns.augment(ns.FieldInt, ns.Field);

```

```
/*
 * @brief Tracks an input field that accepts an IPv4 address.
 *
 * @param[in] id The id attribute of the input field to track.
 */
ns.FieldIAddr =
function (id) {
    this.id = id;
    this.el = document.getElementById(id);
};

ns.FieldIAddr.prototype = {

    /**
     * @brief Validate an IP address string.
     *
     * @returns A valid IP address string (as it was specified in @p value)
     * after removing insignificant zeros and/or leading/trailing white
     * spaces; @c null, on error.
     */
    validate : function(value) {
        var segments, // @p value split on dots
            number, // A single byte of the address
            i;

        value = value + "";
        segments = value.split('.');
        value = '''; // The value is reconstructed
        if(segments.length !== 4) {
            value = null;

        } else {
            /* Convert each segment into a number to validate it. */
            for(i = 0 ; i < segments.length ; ++i) {

                number = parseInt(segments[i], 10);
                if(ns.isNaN(number) || number < 0 || number > 255) {
                    value = null;
                    break;
                }
                /* Join the numbers anew to omit any insignificant zeros. */
                value += number + ".";
            }
        }

        if(i === 4) {
            value = value.substring(0, value.length - 1);
        }
    }
}
```

```

        return value;
    },

    /**
     * @brief Return an array with a single error string.
     *
     * @returns An array of a single string explaining what is a valid IP
     * address.
     */
    _getErrors : function() {
        return ['Απαιτούνται 4 ακέραιοι από 0 μέχρι 255, διαχωρισμένοι με',
            + ' τελεία.'];
    }
};

ns.augment(ns.FieldIAddr, ns.Field);

/**
 * @brief Track a date Field comprised of multiple input fields.
 *
 * Do note, this operates on UTC only (meaning that when setting a date with
 * this.set(), the value for each input element will be extracted via
 * functions such as Date.getUTCFullYear()).
 *
 * To facilitate instantiation, only a single id prefix is required which is
 * assumed to be shared among all input field elements to be tracked by this
 * Field. It is also assumed that, in order to identify each element
 * separately, the following suffixes are:
 * --year
 * --month
 * --date
 * --hours
 * --minutes
 * --seconds
 *
 * The id of this element uses the id of the year (@p idPrefix + "--year").
 *
 * @param[in] idPrefix The common id part of the underlying input fields.
 */
ns.FieldDateGroup =
function (idPrefix) {
    this.id = idPrefix + '--year';
    this.elYear = document.getElementById(idPrefix + '--year');
    this.elMonth = document.getElementById(idPrefix + '--month');
    this.elDate = document.getElementById(idPrefix + '--date');
    this.elHours = document.getElementById(idPrefix + '--hours');
    this.elMinutes = document.getElementById(idPrefix + '--minutes');
    this.elSeconds = document.getElementById(idPrefix + '--seconds');
}

```

```
this.el = this.elYear;
};

ns.FieldDateGroup.prototype = {

    /**
     * @brief Update the Field with the supplied date, provided it is valid.
     *
     * Do note, this sets the underlying fields to the UTC-equivalent of the
     * supplied date.
     *
     * @param[in] blob Either a @c String containing the year or a @c Date
     * object, in which case, the remaining arguments are ignored.
     * @param[in] month The month (@c 1 through @c 12).
     * @param[in] date The date (day of month). If the value supplied
     * evaluates to @c 0, it is set to @c 1 to avoid date underflow due to
     * an empty string.
     * @param[in] hours The hours.
     * @param[in] minutes The minutes.
     * @param[in] seconds The seconds.
     * @returns A Date instance, if a valid date could be constructed from
     * the arguments; @p null, otherwise.
     */
    set : function(blob, month, date, hours, minutes, seconds) {
        var dt = this.validate(blob, month, date, hours, minutes, seconds);

        if(dt !== null) {
            /* Make sure numbers are two digits long (except for year). */
            this.elYear.value = dt.getUTCFullYear();
            this.elMonth.value = ns.fixInt(dt.getUTCMonth() + 1, 2);
            this.elDate.value = ns.fixInt(dt.getUTCDate(), 2);
            this.elHours.value = ns.fixInt(dt.getUTCHours(), 2);
            this.elMinutes.value = ns.fixInt(dt.getUTCMinutes(), 2);
            this.elSeconds.value = ns.fixInt(dt.getUTCSeconds(), 2);
        }
        return dt;
    },

    get : function(errors) {
        var dt = this.set(this.elYear.value,
                          this.elMonth.value,
                          this.elDate.value,
                          this.elHours.value,
                          this.elMinutes.value,
                          this.elSeconds.value);

        if(dt === null && typeof errors === "object") {
            errors[this.id] = this._showErrors();
        }
    }
}
```

```

        return dt;
    },

    reset : function() {
        this.elYear.value = "";
        this.elMonth.value = "";
        this.elDate.value = "";
        this.elHours.value = "";
        this.elMinutes.value = "";
        this.elSeconds.value = "";
        return this;
    },

    /**
     * @brief Validate the arguments as if there were a date.
     *
     * @param[in] blob Either a @c String containing the year or a @c Date
     * object, in which case, the remaining arguments are ignored.
     * @param[in] month The month (@c 1 through @c 12).
     * @param[in] date The date (day of month). If the value supplied
     * evaluates to @c 0, it is set to @c 1 to avoid date underflow due to
     * an empty string.
     * @param[in] hours The hours.
     * @param[in] minutes The minutes.
     * @param[in] seconds The seconds.
     * @returns A Date instance, if a valid date could be constructed from
     * the arguments; @p null, otherwise.
     */
    validate : function(blob, month, date, hours, minutes, seconds) {
        var dt; // The parsed date

        if(typeof blob === "string") {

            /* Month is zero-based. */
            if(!ns.isNaN(month) && month > 0) --month;

            /* Avoid date underflow to previous month. */
            if(date == 0) date = 1;

            /* The device operates in UTC. Assume the values inserted are in
             * UTC as well. */
            dt = new Date(Date.UTC(blob,
                month,
                date,
                hours,
                minutes,
                seconds));
        }
    }
}

```

```
    } else {
        dt = blob;
    }

    /* Check validity of date. For instance, if non-digits where
     * supplied, that would have caused an invalid date (NaN). */
    if(ns.isNaN(dt.getTime())) return null;
    return dt;
},

/**
 * @brief Return an array with a single error string.
 *
 * @returns An array of a single string explaining that the date is
 * invalid.
 */
_getErrors : function() {
    return ['Μη Εγκυρη Ημερομηνία Ημέρα/.'];
}

};

ns.augment(ns.FieldDateGroup, ns.Field);

/** 
 * @brief Validate coordinate fields.
 *
 * Field values are updated to match the one parsed.
 *
 * This actually envelopes three FieldInt members which may be accessed at
 * any time to alter their behaviour. For instance, to set the minimum and
 * maximum allowable values, this.fieldX, this.fieldY and this.fieldZ may be
 * used (provided they have been initialised; see parameters). Defaults are
 * @c 0 and @c 255, respectively. To cancel the effects of a limit to any of
 * the aforementioned Field members, its respective @c min and/or @c max
 * should become undefined (with @c delete or be set to @c undefined).
 *
 * At least one field id should be specified (but if only one is required, it
 * would be more efficient to simply use 'FieldInt' directly). this.id will
 * be set to any one valid supplied arguments, with an order of preference:
 * idX, idY, idZ.
 *
 * When validating, only if each initialised Field validates separately, will
 * this Field succeed as well, in which case, an object with member
 *
 * @param[in] idX The id attribute of the field containing the X-coordinate.
 * Supplying this will initialise this.fieldX. Optional.
 * @param[in] idY The id attribute of the field containing the Y-coordinate.
 * Supplying this will initialise this.fieldY. Optional.
```

```

* @param[in] idZ The id attribute of the field containing the Z-coordinate.
* Supplying this will initialise this.fieldZ. Optional.
*/
ns.FieldPoint =
function (idX, idY, idZ) {
    var id;

    /* Initialise a field for each defined argument. */
    idZ && (this.fieldZ = new ns.FieldInt(idZ, 0, 255)) && (this.id = idZ);
    idY && (this.fieldY = new ns.FieldInt(idY, 0, 255)) && (this.id = idY);
    idX && (this.fieldX = new ns.FieldInt(idX, 0, 255)) && (this.id = idX);

    if(this.id === idX) this.el = this.fieldX.el;
    else if(this.id === idY) this.el = this.fieldY.el;
    else if(this.id === idZ) this.el = this.fieldZ.el;

};

ns.FieldPoint.prototype = {

    /**
     * @brief
     *
     * Because not all three coordinates may have been initialised, it is up
     * to the caller to provide values for those that have been. The others
     * are always ignored. In order for this.set() to succeed, all the
     * initialised Fields must pass validation.
     *
     * @param[in] x Value for this.fieldX.
     * @param[in] y Value for this.fieldY.
     * @param[in] z Value for this.fieldZ.
     * @returns An object with three members at most (@c x, @c y and @c z),
     * each set to the corresponding argument value, if the initialised
     * Fields pass validation; @c null, otherwise.
     */
    set : function(x, y, z) {
        var point = this.validate(x, y, z);

        if(point !== null) {
            /* Since validation succeeded on all of them as a whole, no need
             * to validate each separately, so by-pass .validate(). */
            this.fieldX && (this.fieldX.el.value = point.x);
            this.fieldY && (this.fieldY.el.value = point.y);
            this.fieldZ && (this.fieldZ.el.value = point.z);
        }
        return point;
    },
}

/**

```

```
* @brief Retrieve and validate Field contents.  
*  
* It return @c null, if any of the initialised Fields fails validation.  
* On the contrary, if all initialised Fields succeed, a single object  
* containing a member-value pair for each initialised Field *only*, is  
* returned. If it is required to determine the Field(s) that failed  
* validation, three status members are available: this.isValidX,  
* this.isValidY, this.isValidZ.  
*  
* @param[out] error Contains error strings. If any errors occurred  
* during parsing, they will be inserted into 'error[id]' as an array  
* of strings, where @c id is the first non-empty constructor-supplied  
* value. A single error string is produced for each coordinate similar  
* to the one generated by FieldInt._getError, but adjusted to include  
* the name of that coordinate.  
*/  
get : function(errors) {  
    /* Fetch the values directly from the input fields so that they may  
     * be validated as a group via this.validate(). */  
    var x = this.fieldX ? this.fieldX.el.value : null,  
        y = this.fieldY ? this.fieldY.el.value : null,  
        z = this.fieldZ ? this.fieldZ.el.value : null,  
        point = this.set(x, y, z);  
  
    if(point === null && typeof errors === 'object') {  
        errors[this.id] = this._showErrors();  
    }  
  
    return point;  
},  
  
reset : function() {  
    this.fieldX && (this.fieldX.el.value = ''');  
    this.fieldY && (this.fieldY.el.value = ''');  
    this.fieldZ && (this.fieldZ.el.value = ''');  
},  
  
/**  
 * @brief Validate the supplied values.  
*  
* Each supplied value is validated according to the constraints of the  
* appropriate initialised Field.  
*  
* @returns An object with a member for each initialised Field and the  
* corresponding argument value, if the supplied values pass  
* validation; @c null, otherwise.  
*/  
validate : function(x, y, z) {
```

```

var point = {}, // Contains the parsed values
    value = point; // It may be altered to @c null

/* Assume the values are correct. */
this.isValidX = this.isValidY = this.isValidZ = true;

/* Use .validate() on each initialised Field. @c point is added a
 * member for each such Field and is given the return value of
 * .validate() even if it is @c null, in which case @c value is set
 * to @c null and the corresponding this.isValid[X|Y|Z] is set to @c
 * false. */
this.fieldX && ((point.x = this.fieldX.validate(x)) === null)
    && ((this.isValidX = false) || (value = null));
this.fieldY && ((point.y = this.fieldY.validate(y)) === null)
    && ((this.isValidY = false) || (value = null));
this.fieldZ && ((point.z = this.fieldZ.validate(z)) === null)
    && ((this.isValidZ = false) || (value = null));

/* If there has been at least one error, @c value will no longer be
 * a reference to @c point */
return value;
},

_getErrors : function() {
    var errors = [],
        str;

    if(this.isValidX === false) {
        str = this.fieldX._getErrors()[0];
        str = 'Γιατού X' + str.substring(1);
        errors.push(str);
    }
    if(this.isValidY === false) {
        str = this.fieldY._getErrors()[0];
        str = 'Γιατού Y' + str.substring(1);
        errors.push(str);
    }
    if(this.isValidZ === false) {
        str = this.fieldZ._getErrors()[0];
        str = 'Γιατού Z' + str.substring(1);
        errors.push(str);
    }

    return errors;
}
};

ns.augment(ns.FieldPoint, ns.Field);

```

```
}(window.gNS = window.gNS || {}));
```

## Κώδικας ιστοσελίδας

ui/index.html

### Δυναμική σελίδα «Ρυθμίσεις»

ui/page-config.js

```
; (function (ns) {  
  
    /**  
     * @brief Responsible for rendering the configuration of the device.  
     *  
     * Functions of immediate interest are init(), reload(), submit() and  
     * reset(). init() should be called first, before attempting to operate this.  
     */  
    ns.PageConfig = ns.PageConfig || (function () {  
  
        /* These must *all* be supplied via init(). */  
        var fIAddr,  
            fGateway,  
            fSubnet,  
            fDate,  
            fCoords,  
            eIntervalHrs,  
            eIntervalMins,  
            fSamples;  
  
        var isInternal; /* Denotes whether a request has been sent to  
                         * the device for the needs of the UI and not  
                         * explicitly by the user. */  
  
        /**  
         * @brief Initialise this instance.  
         *  
         * @param[in] s Object with initialisation settings. It should contain  
         * the following keys, each with a value as follows:@verbatim {  
         * form.idIAddr id of iaddr input field  
         * .idGateway id of gateway input field  
         * .idSubnet id of subnet input field  
         * .idDate id prefix (see FieldDateGroup()) of server-date  
         * .idConfigX id of maximum X input field  
         * .idConfigY id of maximum Y input field  
         * .idConfigZ id of maximum Z input field  
         * .idIntervalHrs id of interval hours field  
         * .idIntervalMins id of interval minutes field  
         * .idSamples id of automated samplings field  
         */  
    });  
});
```

```

* .clsError class of 'ul's when displaying field errors
*/@endverbatim
*/
var init = function (s) {
    var cls = s.form.clsError,
        x = s.form.idConfigX,
        y = s.form.idConfigY,
        z = s.form.idConfigZ,
        hours = s.form.idIntervalHrs,
        mins = s.form.idIntervalMins,
        samples = s.form.idSamples;

    fIAddr = (new ns.FieldIAddr(s.form.idIAddr))
        .setErrorClass(cls);
    fGateway = (new ns.FieldIAddr(s.form.idGateway))
        .setErrorClass(cls);
    fSubnet = (new ns.FieldIAddr(s.form.idSubnet))
        .setErrorClass(cls);
    fDate = (new ns.FieldDateGroup(s.form.idDate))
        .setErrorClass(cls);
    fCoords = (new ns.FieldPoint(x, y, z))
        .setErrorClass(cls);
    fSamples = (new ns.FieldInt(samples, 0, 100)
        .setErrorClass(cls));
    elIntervalHrs
        = document.getElementById(hours);
    elIntervalMins
        = document.getElementById(mins);

    /* Minimum operational range values. */
    fCoords.fieldX.min = 1;
    fCoords.fieldY.min = 1;
    fCoords.fieldZ.min = 2;
};

/**
 * @brief Update the view of the Page.
 *
 * The default behaviour is to load the current configuration from the
 * device, replacing anything that may have been set on a previous visit.
 * This is done to ensure that the displayed configuration is up-to-date.
 *
 * @param[in] evt If instance of Event, .preventDefault() will be
 * invoked. Optional.
 */
var reload = function (evt) {
    var req = ns.createRequest();
    window.document.title = "ΤΑΙΟΛικνού-Πυθμίσεις";
}

```

```
evt instanceof Event && evt.preventDefault();

if(!req) return;

reset();

/* Send an erroneous value for the operational range. This results
 * in the device sending its physical limits. Those limits will be
 * used as constraints to the input fields. @c isInternal designates
 * this nature of the following response so that the back-end may act
 * accordingly. */
isInternal = true;

req.open('PUT', 'configuration.php');
req.onreadystatechange = handlePUT;
req.send('{\'x\':9999}');
};

/***
 * @brief Send new configuration to the device.
 *
 * @param[in] evt If instance of Event, .preventDefault() will be
 * invoked. Optional.
 */
var submit = function (evt) {
    var errors = {},
        payload = {}, // Request
        value, // Each parsed value
        req,
        date, // Field value
        coords; // Field parameter value

    evt instanceof Event && evt.preventDefault();

    /* Clear any previously set field messages. */
    resetMsg();

    /* Only include Fields that have been set. */

    if(fIAddr.el.value && (value = fIAddr.get(errors)) !== null) {
        payload.iaddr = value;
    }

    if(fGateway.el.value && (value = fGateway.get(errors)) !== null) {
        payload.gateway = value;
    }
}
```

```
if(fSubnet.el.value && (value = fSubnet.get(errors)) !== null) {
    payload.subnet = value;
}

if(fDate.elYear.value && fDate.elMonth.value && fDate.elDate.value
&& fDate.elHours.value && fDate.elMinutes.value) {
    date = fDate.get(errors);
} else {
    fDate.reset();
}

(fCoords.fieldX.el.value
|| fCoords.fieldY.el.value
|| fCoords.fieldZ.el.value)
&& (value = fCoords.get(errors)) === null || (coords = value);

if(date) {
    payload.date = date.toJSON();
    payload.day = date.getDay();
}
if(coords) {
    payload.x = coords.x;
    payload.y = coords.y;
    payload.z = coords.z;
}

/* Each hours contains 10 quanta; each quantum equals 6 minutes. */
payload.interval = elIntervalHrs.selectedIndex * 10
    + elIntervalMins.selectedIndex;

if(fSamples.el.value && (value = fSamples.get(errors)) !== null) {
    payload.samples = value;
}

/* Display error messages and inform the configuration has not been
* saved. */
if(!ns.isEmpty(errors)) {

} else if(!ns.isEmpty(payload)) {

    req = ns.createRequest();
    req.open('PUT', 'configuration.php');
    req.onreadystatechange = handlePUT;
    req.send(JSON.stringify(payload));
} else {

    /* All fields were empty. Inform user no operation was
     * performed. */
}
```

```
ns.log('Συμπληρώστε πρώτα τα επιθυμητά πεδία και μετά πατήστε Επόμενη');
+ "&laquo;Αποθήκευση;&raquo;;", "critical");
}

};

/***
 * @brief Reset all Fields (including error messages).
 *
 * If it is desired to reset to the current configuration, reload should
 * be used, instead.
 *
 * @param[in] evt If instance of Event, .preventDefault() will be
 * invoked. Optional.
 */
var reset = function (evt) {
    evt instanceof Event && evt.preventDefault();

    resetMsg();
    fIAddr.reset();
    fGateway.reset();
    fSubnet.reset();
    fDate.reset();
    fCoords.reset();
    fSamples.reset();
    elIntervalHrs.selectedIndex = 0;
    elIntervalMins.selectedIndex = 0;
};

/***
 * @brief Request that all Fields remove their error messages.
 */
var resetMsg = function() {
    fIAddr.resetMsg();
    fGateway.resetMsg();
    fSubnet.resetMsg();
    fDate.resetMsg();
    fCoords.resetMsg();
    fSamples.resetMsg();
};

/***
 * @brief Called via onreadystatechange. Handles asynchronous GET.
 *
 * This Page uses method GET only to retrieve the current configuration
 * of the device.
 */
var handleGET = function () {
```

```

if(this.readyState !== 4) return;

if(this.status === 200) loadFields(JSON.parse(this.responseText));
};

/** 
 * @brief Called via onreadystatechange. Handles asynchronous PUT.
 *
 * This Page uses method PUT to update the device configuration. It also
 * uses it, behind the scenes, to retrieve the physical limits of the
 * device and set them as constraints on the Operational Range Field. To
 * do so, it send an erroneous request (which is, of course dropped by
 * the device) but results in the device advertising those limits.
 */
var handlePUT = function () {
    var response;

    if(this.readyState !== 4) return;

    response = JSON.parse(this.responseText);

    if(this.status === 200) {
        /* Inform configuration has been saved. Also, reload the
         * configuration from the device response. */
        ns.log('Η ρυθμίσεις αποθηκεύτηκε.', 'info');
        loadFields(response);
    } else if(this.status === 400) {

        /* If the request was sent to retrieve the absolute physical
         * limits, set them as constraints to the corresponding input
         * fields (@c fConfigX, @c fConfigY and @c fConfigZ). */
        if(isInternal) {
            /* Set current constraints. */
            fCoords.fieldX.max = response.x;
            fCoords.fieldY.max = response.y;
            fCoords.fieldZ.max = response.z;

            /* Reuse this request instance, this time, to actually
             * retrieve the current configuration. */
            this.open('GET', 'configuration.php');
            this.onreadystatechange = handleGET;
            this.send();
        } else {
            console.log('request has been dropped:', response);
            /* Inform request has been dropped, probably due to
             * an erroneous value. Also inform what are the physical
             * limits. */
        }
    }
}

```

```

ns.log('Η συσκευή παρέβριψε το αίτημα λόγω εσφαλμένης παραμήκης.'
+ ' Το λειτουργικό σύστημα μπορεί να ξεπερνά τις'
+ ' φυσικές διαστάσεις της συσκευής. Μια μέγιστης'
+ ' αποδεκτές μέτρες [X, Y, Z] είναι ['
+ response.x + ', ' + response.y + ', '
+ response.z + '].', 'fatal');
}

isInternal = false;
};

/** 
 * @brief Set the fields to the specified values.
 *
 * @param[in] conf An object containing the device configuration, as
 * returned by the corresponding endpoint.
 */
var loadFields = function (conf) {
    var errors = 0,
        interval;

    reset(); // Clear all fields

    fIAddr.set(conf.iaddr) || ++errors;
    fGateway.set(conf.gateway) || ++errors;
    fSubnet.set(conf.subnet) || ++errors;
    fDate.set(new Date(conf.date)) || ++errors;
    fCoords.set(conf.x, conf.y, conf.z) || ++errors;
    fSamples.set(conf.samples) || ++errors;

    interval = conf.interval;
    elIntervalHrs.selectedIndex = Math.floor(interval / 10);
    elIntervalMins.selectedIndex = Math.floor(interval % 10);

    /* TODO: Show message on the unlikely event of an erroneous value */
};

return {"init" : init,
        "reload" : reload,
        "submit" : submit,
        "reset" : reset};
})();
})(window.gNS = window.gNS || {});

```

### Δυναμική σελίδα «Χειρισμός»

ui/page-operate.js

```
; (function (ns) {

    /**
     * @brief Responsible for operating the device manually.
     */
    ns.PageOperate = ns.PageOperate || (function () {

        /* These must *all* be supplied via init(). */
        var elStatus,
            elRange,
            fCoords;

        /**
         * @brief Initialise this instance.
         *
         * @param[in] s Object with initialisation settings. It should contain
         * the following keys, each with a value as follows:@verbatim {
         * form.idNewX id of field for the new X coordinate of device
         * .idNewY id of field for the new Y coordinate of device
         * .clsError class of 'ul's when displaying field errors
         * data.idStatus id of element to display current device state
         * .idRange id of element to display the operational range
         *}@endverbatim
         */
        var init = function (s) {
            var cls = s.form.clsError,
                x = s.form.idNewX,
                y = s.form.idNewY;

            elStatus
                = document.getElementById(s.data.idStatus);
            elRange = document.getElementById(s.data.idRange);
            fCoords = (new ns.FieldPoint(x, y)).setErrorClass(cls);
        };

        /**
         * @brief Update the view of the Page.
         *
         * The default behaviour is to load the current state of the device
         * (position, mostly), while maintaining any previously set coordinates
         * in the input fields.
         *
         * @param[in] evt If instance of Event, .preventDefault() will be
         * invoked. Optional.
         */
        var reload = function (evt) {
            var req1 = ns.createRequest(),
                req2 = ns.createRequest();
```

```
window.document.title = 'ΓΑΙΟΛΙΚΝΟ-ΛΧΕΙΡΙΣΜÓS';

evt instanceof Event && evt.preventDefault();

if(!req1 || !req2) return;

reset();

/* Request current device position. */
req1.open('GET', 'coordinates.php');
req1.onreadystatechange = handleGETCoords;
req1.send();

/* Request operational range. */
req2.open('GET', 'configuration.php');
req2.onreadystatechange = handleGETConfig;
req2.send();
};

/**
 * @brief Update device head position.
 *
 * @param[in] evt If instance of Event, .preventDefault() will be
 * invoked. Optional.
 */
var move = function (evt) {
    var errors = {},
        req,
        coords; // Field parameter value

    evt instanceof Event && evt.preventDefault();
    /* Clear any previously set field messages. */
    resetMsg();

    req = ns.createRequest();
    if(!req) return;

    coords = fCoords.get(errors);

    /* TODO: Display error messages. */
    if(!ns.isEmpty(errors)) {

    } else {
        req.open('PUT', 'coordinates.php');
        req.onreadystatechange = handlePUTCoords;
        req.send(JSON.stringify(coords));
    }
};
```

```
/**  
 * @brief Request the device takes a new measurement.  
 *  
 * @param[in] evt If instance of Event, .preventDefault() will be  
 * invoked. Optional.  
 */  
var sample = function(evt) {  
    var req;  
  
    evt instanceof Event && evt.preventDefault();  
  
    /* Clear any previously set field messages. */  
    resetMsg();  
  
    req = ns.createRequest();  
    if(!req) return;  
    req.open("POST", "measurement.php");  
    req.onreadystatechange = handlePOSTSample;  
    req.setRequestHeader("Content-Length", 0);  
    req.send();  
};  
  
/**  
 * @brief Reset all Fields (including error messages).  
 *  
 * @param[in] evt If instance of Event, .preventDefault() will be  
 * invoked. Optional.  
 */  
var reset = function (evt) {  
    evt instanceof Event && evt.preventDefault();  
  
    resetMsg();  
    fCoords.reset();  
};  
  
/**  
 * @brief Request that all Fields remove their error messages.  
 */  
var resetMsg = function() {  
    fCoords.resetMsg();  
};  
  
/**  
 * @brief Update operational range display and constraints.  
 *  
 * @param[in] conf Object containing @c X and @c Y maximum values.  
 */
```

```
var updateRange = function(conf) {
    fCoords.fieldX.max = conf.x - 1; // Zero-based
    fCoords.fieldY.max = conf.y - 1; // Zero-based
    elRange.innerHTML = conf.x + ", " + conf.y;
};

/**
 * @brief Called via onreadystatechange. Handles GETting configuration.
 *
 * The configuration is used to determine the operational range of the
 * device, that is, the maximum allowable @c X, @c Y, @c Z values. @c Z
 * cannot be set manually; it is either submerged or not.
 */
var handleGETConfig = function () {
    var conf;

    if(this.readyState !== 4) return;

    /* Set field constraints and display operational range. */
    if(this.status === 200) {
        conf = JSON.parse(this.responseText);
        updateRange(conf);
    }
};

/**
 * @brief Called via onreadystatechange. Handles GETting coordinates.
 *
 * The current device coordinates are displayed on the UI for reference.
 * A @c Z value of @c 0 indicates the head is submerged.
 */
var handleGETCoords = function () {
    var conf;

    if(this.readyState !== 4) return;

    /* Load current coordinates. */
    if(this.status === 200) {
        conf = JSON.parse(this.responseText);

        /* By-pass constraints; they may have been altered. */
        fCoords.fieldX.el.value = conf.x;
        fCoords.fieldY.el.value = conf.y;

        elStatus.innerHTML = conf.z === 0
            ? "Δειγματοληψία"
            : "σετοιμότητα";
    }
};
```

```

} else if(this.status === 503) {
    elStatus.innerHTML = "εντοκινήσει";

    /* Display estimated time of completion. */
    ns.log("Η συσκευή δήλωσε απασχολημένη. Εκτιμάμενος χρόνος"
        + " πολοκλήρωσης τρέχουσας εργασίας:" +
        + ns.minsec(this.getResponseHeader('Retry-After')),
        "info");
}

/** 
 * @brief Called via onreadystatechange. Handles PUTting coordinates.
 *
 * This Page uses method PUT to update the device position.
 */
var handlePUTCoords = function () {
    var response;

    if(this.readyState !== 4) return;

    switch(this.status) {
        /* The device already is at the requested position. */
        case 200:
            response = JSON.parse(this.responseText);
            ns.log("Η συσκευή αποκρίθηκε ότι ο βρισκεται στη θέση"
                + "[X,Y] [" + response.x + ", " + response.y + "]",
                "info");
            break;

        /* Header Retry-After designates how long it should take to
         * reach the specified position. */
        case 202:
            ns.log("Η συσκευή αποδέχθηκε το αίτημα. Εκτιμάμενος χρόνος"
                + " πολοκλήρωσης:" +
                + ns.minsec(this.getResponseHeader('Retry-After')),
                "info");
            break;

        /* Bad request. Display operational range; maybe they have been
         * modified in-between requests. */
        case 400:
            response = JSON.parse(this.responseText);
            ns.log("Η συσκευή απέρριψε το αίτημα λόγω εσφαλμένης τιμής."
                + " Το υπέβαλλε η συσκευή σε δηλώθηκε στις είναι"
                + "[X,Y] [" + response.x + ", " + response.y + "]",
                "fatal");
            updateRange(response);
    }
}

```

```

break;

/* Device is busy. */
case 503:
    ns.log('Η συσκευή δήλωσε απασχόλημένη. Εκτιμάμενος χρόνος',
           + ' ολοκλήρωσης τρέχουσας εργασίας: ' +
           + ns.minsec(this.getResponseHeader('Retry-After')),
           + 'info');
    break;
}

};

/** 
 * @brief Called via onreadystatechange. Handles POSTing a measurement.
 *
 * This Page uses method POST to request taking a new sample.
 */
var handlePOSTSample = function () {
    if(this.readyState !== 4) return;

    switch(this.status) {

        /* Header Retry-After designates how long it should take to
         * complete the request. */
        case(202):
            ns.log('Η συσκευή αποδέχθηκε το αίτημα. Εκτιμάμενος χρόνος',
                   + ' ολοκλήρωσης: ' +
                   + ns.minsec(this.getResponseHeader('Retry-After')),
                   + 'info');
            break;

        /* The device is busy. */
        case(503):
            ns.log('Η συσκευή δήλωσε απασχόλημένη. Εκτιμάμενος χρόνος',
                   + ' ολοκλήρωσης τρέχουσας εργασίας: ' +
                   + ns.minsec(this.getResponseHeader('Retry-After')),
                   + 'info');
            break;
    }
};

return {'init' : init,
        'reload' : reload,
        'move' : move,
        'sample' : sample};
})();
}) (window.gNS = window.gNS || {});

```

## Αρχείο μορφοποιήσεων

### ui/style.css

```
body {  
    background: #DDD;  
    background: repeating-linear-gradient(to right, #CCC, #FFF, #CCC);  
    font-family: 'MgOpen-Cosmetica', 'Arial Unicode MS';  
    margin: 8px;  
    color: #404040;  
}  
  
select > option {  
    font-family: 'MgOpen-Cosmetica', 'Arial Unicode MS'  
}  
  
.button {  
    font-family: 'MgOpen-Cosmetica', 'Arial Unicode MS';  
}  
  
h1 {  
    margin: 0.2em 0;  
    font-size: 120%;  
}  
  
h2 {  
    font-size: 110%;  
}  
  
h3 {  
    font-size: 100%;  
}  
  
h4 {  
    font-size: 90%;  
}  
  
p {  
    margin: 0.3em;  
}  
  
a {  
    color: inherit;  
    text-decoration: none;  
    border-bottom: 1px dotted #000;  
    line-height: 100%;  
    display: inline-block;  
}
```

```
a:visited {  
    color: inherit;  
}  
  
tr {  
    line-height: 150%;  
}  
  
th, td {  
    padding: 0 0.3em;  
    vertical-align: top;  
}  
  
#container {  
    min-width: 500px;  
    min-height: 15em;  
    max-width: 850px;;  
  
    margin: auto;  
    background-color: white;  
    box-shadow: 0 0 3px 1px #AAA;  
    border-radius: 3px;  
}  
  
#header {  
    padding: 5px;  
}  
#header > * {  
    vertical-align: middle;  
}  
  
#logo {  
    background-image: url('logo.png');  
    width: 80px;  
    height: 80px;  
    margin: 10px;  
    display: inline-block;  
}  
#logo-text {  
    display: inline-block;  
}  
  
#nav {  
    border-top: 2px solid #DDD;  
    border-bottom: 2px solid #DDD;  
  
    margin: 0;  
    padding: 0 0 0.5em 0;
```

```
padding: 0;

text-align: center;
background: #EEE;
}

#nav > li {
    display: inline-block;
    margin: 0 0.2em;
    transition: background 0.2s;
}
#nav > li.current {
    background: #DDD;
}

#nav > li:hover {
    background: #DDD;
}

#nav a {
    border: none;
    text-decoration: none;
    outline: none;
    display: block;
    padding: 0.2em 0.7em;
    line-height: 110%;
}

#content {
    margin: 1em;
    transition: opacity 0.2s;
    min-height: 3em;
}
#content p {
    font-size: 90%;
    line-height: 130%;
}

#infobox {
    margin: 0;
    padding: 5px;
    color: #777;
    border-top: 1px solid #DDD;
    overflow: auto;
    height: 4em;
    transition: height 1s 0.7s;
    background: #EEE;
}
```

```
#infobox:hover {  
    color: #000;  
    height: 10em;  
}  
  
#infobox > div {  
    margin: 0.2em 0 0.2em 0;  
    padding: 0.1em 0.3em;  
    transition: background 1s;  
    border-radius: 3px;  
}  
  
#infobox > h2 {  
    margin-top: 0.2em;  
    margin-bottom: 0.2em;  
}  
  
.inline {  
    display: inline;  
}  
  
.visible {  
    display: inherit;  
}  
  
.hidden {  
    display: none;  
}  
  
.right {  
    text-align: right;  
}  
  
.small {  
    font-size: small;  
}  
  
.faded {  
    opacity: 0;  
}  
  
.emerged {  
    opacity: 1;  
}  
  
.important {  
}  
  
.tiny {  
    width: 1.5em;
```

```
}

.short {
    width: 2em;
}
.medium {
    width: 2.5em;
}
.big {
    width: 10em;
}

table.log {
    width: 100%;
    text-align: center;
}
table.log td {
    font-family: mono;
    font-size: small;
}
table.log th:nth-child(1), td:nth-child(1) {
    text-align: right;
}
table.log th:nth-child(2), td:nth-child(2) {
    text-align: left;
}
table.log td:nth-child(3) {
    text-align: right;
}
table.log td:nth-child(4) {
    text-align: left;
}

table.stripes {
    border-collapse: separate;
    border-spacing: 0;
    border-top: 3px solid #CCC;
    border-bottom: 3px solid #CCC;
    border-left: 1px solid #CCC;
    border-right: 1px solid #CCC;
    border-radius: 3px;
}
table.stripes thead tr:nth-child(1) {
    background: #CCC;
}
table.stripes tbody tr:nth-of-type(2n-1) {
    background: #EEE;
}
table.stripes tbody tr {
```

```
    transition: background 0.2s;
}
table.stripe tbody tr:hover {
    background: #DDD;
}

label {
    display: block;
    font-size: 90%;
}

.pagination-wrapper {
    margin: 0.5em 0 0.2em;
}

.pagination > a {
    text-align: center;
    margin: 0.2em 0.2em;
    width: 1.2em;
    padding: 0.2em 0.1em;
}

.textfield {
    padding: 0;
}

.button.current {
    background: #DDD;
    cursor: auto;
}

.button {
    padding: 0.2em 1em;
    margin: 0.1em 0.2em;
    display: inline-block;
    background: #EEE;
    border: 1px solid #CCC;
    border-radius: 3px;
    cursor: pointer;
    transition: background 0.2s;
}
.button:hover {
    background: #DDD;
}

.field-wrapper {
    margin: 0.4em 0 0.4em 0;
}
```

```
.more-text {
    max-height: 0;
    overflow: auto;
    transition: max-height 0.2s 0.1s;
}

.more:focus ~ .more-text {
    max-height: 7em;
    overflow: auto;
}

.field-msg {
    font-size: small;
    margin-left: 0.2em;
    vertical-align: top;
}
ul.field-msg {
    padding: 0;
    margin: 0 0 0.2em 0.4em;
    list-style-position:inside;
}
.field-msg.hidden {
    display: none;
}
.field-msg.error {
    color: #B00;
}

.new {
    color: #404040;
}
.fatal.new {
    color: #B00;
    border: 1px solid #B00;
    background: #FDD;
}

.critical.new {
    color: #B08000;
    border: 1px solid #b08000;
    background: #fff0CC;
}

.info.new {
    color: #00B;
    border: 1px solid #00B;
    background: #e0e2e5;
```

}





## Παράρτημα C

### Φωτογραφία



# Βιβλιογραφία

- Albert, Alain (2011). *Understanding CNC Routers*. FPIInnovations. 105 σσ.
- Arduino (2014a). *Arduino Development Environment*. url: <http://arduino.cc/en/Guide/Environment>.
- (2014b). *Arduino Uno*. url: <http://arduino.cc/en/Main/ArduinoBoardUno>.
- (2014c). *Libraries*. url: <http://arduino.cc/en/Guide/Libraries>.
- Atmel Corporation (2004). *AVR318: Dallas 1-Wire® master*. Εχδ. 1. 21 σσ. url: <http://www.atmel.com/devices/ATMEGA328.aspx?tab=documents> (επίσκεψη 07/05/2014).
- (2012a). *8-bit AVR Microcontroller with 8/16/32K Bytes of ISP Flash and USB Controller*. 310 σσ. url: <http://www.atmel.com/devices/atmega16u2.aspx> (επίσκεψη 07/03/2014).
- (2012b). *Atmel AVR911: AVR Open Source Programmer*. 16 σσ. url: <http://www.atmel.com/Images/doc2568.pdf> (επίσκεψη 20/09/2014).
- (2013). *ATmega48A, ATmega48PA, ATmega88A, ATmega88PA, ATmega168A, ATmega1688PA, ATmega328, ATmega328P datasheet*. Εχδ. 1.7. 660 σσ. url: <http://www.atmel.com/devices/ATMEGA328.aspx?tab=documents> (επίσκεψη 01/01/2014).
- Berners-Lee, T., R. Fielding και L. Masinter (1998). *Uniform Resource Identifiers (URI): Generic Syntax*. RFC 2396. RFC Editor. 40 σσ. url: <http://tools.ietf.org/html/rfc2396>.
- (2005). *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. RFC Editor. 61 σσ. url: <http://tools.ietf.org/html/rfc3986>.
- Blender Foundation (2014). *Introduction*. url: <http://wiki.blender.org/index.php/Doc:2.6/Manual/Introduction>.
- Booth, David κ.ά. (2004). *Web Services Architecture*. W3C Note. W3C. url: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211> (επίσκεψη 16/07/2014).
- Doxygen (2014). *Generate documentation from source code*. url: <http://www.stack.nl/~dimitri/doxygen/index.html>.

- Dynamics Research Corporation (1976). *Techniques for Digitizing Rotary and Linear Motion*.
- Fairchild Semiconductor (2002). *Application Note AN-3005. Design Fundamentals for Phototransistor Circuits*. Ежд. 4. 2 σσ. url: <http://www.fairchildsemi.com/an/AN/AN-3005.pdf> (επίσκεψη 30/03/2014).
- Fielding, Roy Thomas (2000). «Architectural Styles and the Design of Network-based Software Architectures». Doctoral dissertation. University of California, Irvine. 162 σσ. url: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- Fielding, R. x.ά. (1999). *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. RFC Editor. 176 σσ. url: <http://www.ietf.org/rfc/rfc2616.txt>.
- Free Software Foundation (2012). «GCC Command Options». Στο: *A GNU Manual*. url: <https://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc/Invoking-GCC.html>.
- Gaillard, Frédéric και Andreas Eieland (2013). *Microprocessor (MPU) or Microcontroller (MCU)?* url: [http://www.atmel.com/Images/MCU\\_vs\\_MPUs\\_Article.pdf](http://www.atmel.com/Images/MCU_vs_MPUs_Article.pdf).
- Gimp (2014). *Introduction*. url: <http://docs.gimp.org/2.8/en/introduction.html>.
- Git (2014). *Getting Started - About Version Control*. url: <http://www.git-scm.com/book/en/v2/Getting-Started-About-Version-Control>.
- Hitec (2002). *General Servo Information*. 2 σσ. url: <http://hitecrcd.com/files/Servomanual.pdf> (επίσκεψη 23/04/2014).
- Inkscape (2014). *Inkscape Overview*. url: <https://inkscape.org/en/about/>.
- Intel Corporation (1988). *Hexadecimal Object File Format Specification*. Ежд. A.
- ISO/IEC (2007). *Programming languages – C*. ISO/IEC 9899:TC3. JTC 1. 540 σσ. url: <http://www.open-std.org/jtc1/sc22/wg14/www/standards>.
- Kuphaldt, Tony R. (2009). *Lessons in Electric Circuits. Semiconductors*. 5η έκδοση. Τόμ. 3. 6 τόμοι. 517 σσ.
- Lynch, Kevin M. και Michael A. Peshkin (2002). «Linear and Rotational Sensors». Στο: *The Mechatronics Handbook 1*.
- Maxim Integrated (2002). *Crystal Considerations for Dallas Real-Time Clocks. Application Note 58*. 8 σσ. url: <http://pdfserv.maximintegrated.com/en/an/AN58.pdf> (επίσκεψη 11/06/2014).
- (2008a). *DS1307. 64 x 8, Serial, I<sup>2</sup>C Real-Time Clock*. 14 σσ. url: <http://datasheets.maximintegrated.com/en/ds/DS1307.pdf> (επίσκεψη 11/06/2014).

- (2008b). *DS18B20 Programmable Resolution 1-Wire® Digital Thermometer*. 27 σσ. url: <http://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html> (επίσκεψη 10/03/2014).
- Microchip Technology (2003). *25AA1024/25LC1024. 1 Mbit SPI™ Bus Serial Flash*. 28 σσ. url: <http://ww1.microchip.com/downloads/en/DeviceDoc/22064D.pdf> (επίσκεψη 05/08/2014).
- Morse, Dennis, Roland McGrath και Mike Frysinger (2014). «Introduction». Στο: *GNU make*. url: [https://www.gnu.org/software/make/manual/html\\_node/index.html](https://www.gnu.org/software/make/manual/html_node/index.html).
- Motorola Inc (2004). *SPI Block Guide*. Έχδ. 04.01. 40 σσ. url: <http://www.engr.colostate.edu/ECE251/S12SPIV4.pdf> (επίσκεψη 10/07/2014).
- Myklebust, Gaute (1997). *The AVR Microcontroller and C Compiler Co-Design*. Atmel Corporation. 6 σσ. url: [http://www.atmel.com/dyn/resources/prod\\_documents/COMPILER.pdf](http://www.atmel.com/dyn/resources/prod_documents/COMPILER.pdf) (επίσκεψη 19/10/2014).
- NXP Semiconductors (2014). *I<sup>2</sup>C-bus specification and user manual*. Έχδ. 6.0. 64 σσ. url: [http://www.nxp.com/documents/user\\_manual/UM10204.pdf](http://www.nxp.com/documents/user_manual/UM10204.pdf) (επίσκεψη 13/06/2014).
- Oetiker, Tobias (2011). *The Not So Short Introduction to L<sup>A</sup>T<sub>E</sub>X 2<sub>E</sub>*. url: <http://www.ctan.org/pkg/lshort-english>.
- OPTEK Inc (2004). *Application Circuits of the Phototransistor Switch. Application Bulletin 213*. 3 σσ. url: <http://www.optekinc.com/pdf/App%20Bulletin%202013-Opto%20Assemblies.pdf> (επίσκεψη 30/03/2014).
- Philips Semiconductors (2003). *I<sup>2</sup>C MANUAL*. 51 σσ. url: [http://www.nxp.com/documents/application\\_note/AN10216.pdf](http://www.nxp.com/documents/application_note/AN10216.pdf) (επίσκεψη 13/06/2014).
- Qucs (2014). *Qucs Project*. url: <http://qucs.sourceforge.net/>.
- Saha, Pradip K (2000). *Aluminum extrusion technology*. ASM International.
- Savannah GNU (2012). *avr-libc*. 416 σσ. url: <http://savannah.nongnu.org/download/avr-libc/avr-libc-user-manual-1.8.0.pdf.bz2>.
- Seames, Warren S (2001). *Computer numerical control: concepts and programming*. Cengage Learning.
- Séquin, Carlo H και David A Patterson (1982). *Design and Implementation of RISC I*. University of California, Berkeley. 25 σσ.
- T. Bray, Ed. (2014). *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. RFC Editor. 16 σσ. url: <http://tools.ietf.org/html/rfc7159>.
- The Santa Cruz Operation (1997). *Application Binary Interface*. 271 σσ. url: <http://www.sco.com/developers/devspecs/gabi41.pdf>.

- Vishay Semiconductors (2002). *Application of Optical Reflex Sensors. TCRT1000, TCRT5000, CNY70*. 17 σσ. url: <http://www.vishay.com/docs/81449/81449.pdf> (επίσκεψη 10/03/2014).
- (2006). *Application of Optical Sensors*. 7 σσ. url: <http://www.vishay.com/docs/80107/80107.pdf> (επίσκεψη 01/03/2014).
- (2009). *TCRT5000, TCRT5000L*. Εξδ. 1.7. 12 σσ. url: <http://www.vishay.com/docs/83760/tcrt5000.pdf> (επίσκεψη 20/02/2014).
- WIZnet (2011). *W5100 Datasheet*. Εξδ. 1.2.4. 71 σσ. url: [http://www.wiznet.co.kr/Sub\\_Modules/en/product/product\\_detail.asp?Refid=653&page=1&cate1=5&cate2=7&cate3=26&pid=1011&cType=2](http://www.wiznet.co.kr/Sub_Modules/en/product/product_detail.asp?Refid=653&page=1&cate1=5&cate2=7&cate3=26&pid=1011&cType=2) (επίσκεψη 25/01/2014).
- (2013). *WIZ811MJ Datasheet*. Εξδ. 1.2. 13 σσ. url: [http://www.wiznet.co.kr/Sub\\_Modules/en/product/product\\_detail.asp?Refid=39&page=1&cate1=&cate2=&cate3=&pid=1030&cType=2#tab](http://www.wiznet.co.kr/Sub_Modules/en/product/product_detail.asp?Refid=39&page=1&cate1=&cate2=&cate3=&pid=1030&cType=2#tab) (επίσκεψη 25/01/2014).
- Wunsch, Joerg (2013). *avrdude – driver program for “simple” Atmel AVR MCU programmer*. url: <http://www.nongnu.org/avrdude/user-manual/avrdude.html>.
- Κοῖλιας, Χρήστος (2004). *Δομές Δεδομένων και Ογγανώσεις Αρχείων*. 2η έκδοση. Εκδόσες Νέων Τεχνολογιών. 447 σσ. isbn: 960-8105-64-1.