ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

Αναφορά

# 8η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Εφαρμογή IoT για νοσοκομείο

(Ο κώδικας παρατίθεται σε ξεχωριστό αρχείο)

**Ομάδα 1**

Ξανθόπουλος Παναγιώτης (03119084)

Παπαναστάσης Αθανάσιος (03113197)

## Ζήτημα 8.1

Αρχικά, χρησιμοποιούμε τις συναρτήσεις για το twi, το PCA9555 και την LCD όπως στην προηγούμενη άσκηση. Έχουμε 3 συναρτήσεις για τον χειρισμό της USART, οι οποίες δίνονται στην εκφώνηση.

Για την λήψη μηνυμάτων, χρησιμοποιούμε έναν buffer μήκους 100 bytes, τον recv_msg_buf. Χρησιμοποιούμε και μια βοηθητική μεταβλητή, την recv_msg_lim, ώστε να γνωρίζουμε πόσες θέσεις του πίνακα πιάνει το τελευταίο μήνυμα που αποθηκεύσαμε σε αυτόν. Την λήψη των μηνυμάτων την πραγματοποιεί η συνάρτηση read_store. Η συνάρτηση αυτή αρχικά μηδενίζει το recv_msg_lim και στη συνέχεια λαμβάνει χαρακτήρες σε ένα αέναο while loop. Κάθε χαρακτήρα που λαμβάνει, τον συγκρίνει με το '\n'. Αν είναι ίδιος, βγαίνει από το loop και επιστρέφει, αλλιώς τον αποθηκεύει στον buffer, αυξάνει το lim και συνεχίζει.

Για την αποστολή των μηνυμάτων, χρησιμοποιούμε την συνάρτηση send_msg. Η συνάρτηση αυτή παίρνει ως ορίσματα έναν πίνακα από char και το μήκος του προς μετάδοση μηνύματος. Σε ένα for loop, μεταδίδει τον έναν χαρακτήρα μετά τον άλλον.

Στη συνέχεια, ορίζουμε 2 συναρτήσεις, την print_success και την print_fail. Αμφότερες παίρνουν όρισμα το step και τυπώνουν «{step}. Success» ή «{step}. Fail». Το step αντιπροσωπεύει το βήμα στο οποίο βρίσκεται η main (περισσότερα στη συνέχεια). Για παράδειγμα, στο πρώτο βήμα (connect), θα τυπώσουμε σε περίπτωση αποτυχίας το «1. Fail».

Ορίζουμε μια συνάρτηση, την check_print, η οποία συγκρίνει το περιεχόμενο του buffer με την συμβολοσειρά "Success" (μαζί με τα "). Αν είναι ίδια, καλεί την print_success, αλλιώς καλεί την print_fail. Παίρνει ως όρισμα πάλι το step, το οποίο περνά στις 2 παραπάνω συναρτήσεις. Επιστρέφει 1 σε περίπτωση fail και 0 σε περίπτωση success.

Τέλος, ορίζουμε 2 συναρτήσεις, την esp_connect και την esp_url, οι οποίες πραγματοποιούν τα 2 πρώτα βήματα του προγράμματος. Η esp_connect καλεί την send_msg με μήνυμα το "ESP:connect\n" (και μήκος το μήκος της συμβολοσειράς αυτής). Στη συνέχεια καλεί την read_store ώστε να ληφθεί και να αποθηκευτεί η απάντηση του ESP. Τέλος, καλεί την check_print με όρισμα το 1 για να ελεγχθεί η απάντηση και να τυπωθεί στην LCD το αντίστοιχο μήνυμα. Τέλος, επιστρέφει ότι επέστρεψε η check_print. Η esp_url λειτουργεί ανάλογα, απλώς μεταδίδει το μήνυμα "ESP:url:\"http://192.168.1.250:5000/data\"\n" και αντιστοιχεί σε step=2.

Στη main αρχικοποιούμε το twi, το usart (με baudrate 9600) και το port expander για την διαχείριση της LCD.

Σε ένα αέναο while καλούμε αρχικά την esp_connect μέχρι να επιτύχει (με όριο προσπαθειών το 2). Στη συνέχεια κάνουμε το ίδιο για την esp_url και επαναλαμβάνουμε.

## Ζήτημα 8.2

Προσθέτουμε τις δοσμένες συναρτήσεις για το one wire interface. Επίσης, προσθέτουμε τις συναρτήσεις scan_row, scan_keypad, scan_keypad_rising_edge και keypad_to_ascii μαζί με τον πίνακα key_array (βλ. άσκηση 6.2). Οι συναρτήσεις αυτές αναλαμβάνουν να διαβάσουν το πληκτρολόγιο (μέσω του Port expander) και να επιστρέψουν τον ASCII χαρακτήρα που πατήθηκε. Αν δεν πατήθηκε κάποιο κουμπί, επιστρέφουν το 0 (NULL).

Για την αποθήκευση της κατάστασης του ασθενή (θερμοκρασία, πίεση, ομάδα και status) χρησιμοποιούμε ένα struct που έχει ως πεδία τα παραπάνω 4 χαρακτηριστικά. Για την ανάθεση τιμών στα πεδία του struct προσθέτουμε τις 4 παρακάτω συναρτήσεις.

Έχουμε την read_temp, η οποία είναι ίδια με αυτήν της άσκησης 7.2. με μερικές αλλαγές. Αν η 16bit τιμή που διαβάζεται από το one_wire είναι η 0x8000 (No device), θέτουμε την θερμοκρασία ίση με 0 για να δείξουμε ότι υπάρχει βλάβη του θερμόμετρου. Αλλιώς, την μετατρέπουμε σε float (όπως στην άσκηση 7.2) και την περνάμε στο πεδίο temperature του struct. Προσθέτουμε πάντα την τιμή 16.3 ώστε η θερμοκρασία που διαβάζεται να προσομοιάζει θερμοκρασία σώματος και όχι δωματίου.

Έχουμε επίσης την adc_init, η οποία αρχικοποιεί το ADC σε single conversion mode.

Για την ανάγνωση της πίεσης, έχουμε την read_pressure, η οποία ξεκινά μια μετατροπή ADC, διαιρεί το αποτέλεσμα με το 1024, το πολλαπλασιάζει με το 20 (ώστε να έχουμε τιμές από 0 μέχρι 20) και το περνά στο πεδίο pressure του struct.

Τέλος, έχουμε την change_status η οποία αναλαμβάνει να αλλάξει το status του ασθενή. Αν βρει πατημένο το κουμπί 1 (νούμερο ομάδας μας), θέτει το status ίσο με NURSECALL και επιστρέφει. Αν η πίεση είναι κάτω από 4 ή πάνω από 12 cm H2O, θέτει το status ίσο με CHECKPRESSURE και επιστρέφει. Αν η θερμοκρασία είναι κάτω από 34 ή πάνω από 37 βαθμούς κελσίου, θέτει το status ίσο με CHECKTEMPERATUREκαι επιστρέφει. Αν βρει πατημένο το κουμπί #, θέτει το status ίσο με OK και επιστρέφει. Τέλος, αν δεν ισχύει κανένα από τα παραπάνω (προφανές εφόσον κάθε ενδεχόμενο επιστρέφει) και αν το status δεν είναι NURSECALL (θέλουμε το NURSECALL να είναι persistent έως ότου είτε υπάρξει πρόβλημα με την πίεση/θερμοκρασία είτε πατηθεί το #), το status γίνεται OK και επιστρέφουμε.

Για την εμφάνιση των παραπάνω στην οθόνη, χρησιμοποιούμε την print_to_lcd, η οποία μορφοποιεί και τυπώνει τα περιεχόμενα του struct στην LCD.

Στην main, προσθέτουμε στις αρχικοποιήσεις την κλήση της adc_init και την αρχικοποίηση του port expander για τον χειρισμό του πληκτρολογίου. Επίσης, προσθέτουμε ως βήματα στο αέναο while τις κλήσεις στην read_temp, read_pressure, change_status και print_to_lcd ώστε να πραγματοποιείται η ανανέωση των τιμών και το τύπωμά τους στην οθόνη.

## Ζήτημα 8.3

Στο ζήτημα αυτό υλοποιούμε την δημιουργία του payload και την μετάδοση αυτού. Προσθέτουμε τις παρακάτω συναρτήσεις.

Αρχικά, έχουμε την parse_payload η οποία αναλαμβάνει να «φτιάξει» το payload string όπως το απαιτεί το ESP. Το πραγματοποιεί μορφοποιώντας τα πεδία του struct και τοποθετώντας τα σε έναν global πίνακα payload_json, με αλλεπάλληλες κλήσεις της snprintf. Ως βοηθητική παράμετρος χρησιμοποιείται ο ακέραιος json_pos (κατ' αντιστοιχία του recv_msg_lim), ο οποίος κρατά μέχρι ποια θέση του πίνακα payload_json έχουμε γράψει ως τώρα.

Την συνάρτηση αυτή καλεί η esp_payload, η οποία υλοποιεί το βήμα 3. Η συνάρτηση αυτή, αφού «φτιάξει» το payload μέσω της parse_payload_json, καλεί την send_msg ώστε να μεταδοθεί ο πίνακας payload_json στο ESP. Κατά τα γνωστά, μετά καλεί την read_store για να λάβει την απάντηση του ESP και μετά καλεί την check_print με step=3.

Τέλος, έχουμε την esp_transmit, η οποία στέλνει το string "ESP:transmit\n" μέσω της send_msg και διαβάζει την απάντηση του ESP. Τώρα όμως θέλουμε να τυπώσουμε απλά το μήνυμα που λάβαμε (το οποίο είναι η απάντηση του server). Για το λόγο αυτό, ορίζουμε νέα συνάρτηση, την print_recvbuf, η οποία τυπώνει στην LCD τους χαρακτήρες «4. ». Στη συνέχεια, τυπώνει όλο το περιεχόμενο του recv_msg_buf στην LCD.

Στην main, αρχικοποιούμε το πεδίο team του struct ίσο με 1. Στο αέναο while, καλούμε μετά από τα βήματα των ζητημάτων 8.1, 8.2 την esp_payload μέχρι να επιτύχει (με όριο προσπαθειών το 2). Στη συνέχεια καλούμε μια φορά την esp_transmit και επαναλαμβάνουμε όλο τον παραπάνω κύκλο.

Παραθέτουμε μόνο τον κώδικα του 8.3 σε ξεχωριστό αρχείο καθώς είναι συνολικός και περιλαμβάνει και τα υπόλοιπα ζητήματα.