Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

8η Σειρά Ασκήσεων

Μάθημα: Εργαστήριο Μικροϋπολογιστών

Εξάμηνο: 7ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Ζητήματα 8.1 - 8.2 - 8.3

Να γραφεί πρόγραμμα σε C για τον ATmega328 στο ntuAboard το οποίο να στέλνει στην UART κατάλληλη εντολή για να συνδεθεί το ESP8266 στο δίκτυο.

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include "keypad.h"
#include "twi-pca.h"
#include "LCD.h"
#include "potentiometers.h"
#include "temperature.h"
#include "usart.h"

char status[20] = "OK";
int nurse_status = 0;
```

```
char payload[1000] = "";

void status(long pressure, long temperature) {
    char pressedKey;
    if (scan_keypad_rising_edge()) {
       pressedKey = keypad_to_ascii();
    }

    if (pressedKey == '0' || nurse_status == 1) {
        strcpy(status, "NURSECALL");
        nurse_status = 1;
    }
    if (pressedKey == '#') {
        strcpy(status, "OK");
        nurse_status = 0;

    } else if ((pressure >= 12 || pressure < 4) && nurse_status == 0)
{
        strcpy(status, "CHECKPRESSURE");
    } else if ((temperature < 34 || temperature >= 37) &&
nurse_status == 0) {
        strcpy(status, "CHECKTEMP");
    } else if (nurse_status == 0) {
        strcpy(status, "OK");
    }
}

int main(void) {

    usart_init(103);
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as
output
    PCA9555_0_write(REG_OUTPUT_0, 0x00);

    LCD_init();
    _delay_ms(100);
```

```
usartcheck1("ESP:restart");

_delay_ms(1500);
LCD_clear_display();

char text[15];

if (usartcheck1("ESP:connect")) {
    strcpy(text, "1.Success\0");
} else {
    strcpy(text, "1.Fail\0");
}

LCD_string(text);

_delay_ms(1500);
LCD_clear_display();

if (usartcheck4("ESP:url:\"http://192.168.1.250:5000/data\"")) {
    strcpy(text, "2.Success\0");
} else {
    strcpy(text, "2.Fail\0");
}

LCD_string(text);

_delay_ms(1000);

LCD_init();
_delay_ms(100);

while (1) {
    LCD_init();
    _delay_ms(1000);
    LCD_clear_display();
    long temperature = 0;
    long IntTemp = 0;
    long DecTemp = 0;
```

```
        uint16_t pressure = get_pressure();
        int16_t rawTemp = get_temperature();
        long final_pressure = pressure * 20;
        final_pressure /= 1023;
        char buffer[32];

        if (rawTemp != 0x8000) {
            temperature = ((long) rawTemp * 625) / 10; // Convert to
3 decimal places

            // Extract integer and decimal parts
            IntTemp = (temperature / 1000) + 12; // Integer part
            DecTemp = abs(temperature % 1000); // Decimal part

            status(final_pressure, temperature / 1000 + 12);

            sprintf(buffer, "T=%02ld.%03ld P=%ld", IntTemp, DecTemp,
final_pressure);
            LCD_string(buffer);
            LCD_command(0xC0);
            _delay_ms(2);
            LCD_string(status);
            _delay_ms(1500);
            LCD_clear_display();

        } else {
            // If no device is found, display "NO DEVICE"
            status(final_pressure, temperature / 1000 + 12);

            sprintf(buffer, "NO DEVICE P=%ld", final_pressure);
            LCD_string(buffer);
            LCD_command(0xC0);
            _delay_ms(2);
            LCD_string(status);
            _delay_ms(1500);
            LCD_clear_display();
        }

        sprintf(payload, "ESP:payload:[{\"name\" : \"temperature\",
```

```
\"value\" : \"%03ld.%03ld\"},"
                "{\"name\" : \"pressure\", \"value\" : \"%ld\"},"
                "{\"name\" : \"team\", \"value\": \"30\"},"
                "{\"name\": \"status\",\"value\": \"%s\"}]",
                IntTemp, DecTemp, final_pressure, status);

        usart_init(103);
        char *pay;
        pay = payload;

        if (usartcheck2(pay)) {
            strcpy(text, "3.Success");
        } else {
            strcpy(text, "3.Fail");
        }

        LCD_string(text);

        _delay_ms(1000);
        LCD_clear_display();
        _delay_ms(100);

        usartcheck3("ESP:transmit");
        _delay_ms(1000);


    }
}
```

Αρχικά, στον κώδικα, κάνουμε initialize το USART, Two-Wire Interface και την LCD οθόνη.

Στο πρώτο μέρος του κώδικα, στέλνουμε στην UART κατάλληλη εντολή για να συνδεθεί το ESP8266 στο δίκτυο. Στη συνέχεια, διαβάζουμε από την UART και περιμένουμε την απάντηση του ESP8266 και αν αυτή είναι "Success" απεικονίζουμε στην LCD το μήνυμα 1.Success ενώ αν είναι "Fail" τυπώνουμε στην LCD το μήνυμα 1.Fail . Επαναλαμβάνουμε το ίδιο για την εντολή url ώστε να θέσουμε το url σε http://192.168.1.250:5000/data και αντίστοιχα να εμφανίζουμε μηνύματα 2.Success και 2.Fail.

Για τον έλεγχο των απαντήσεων που λαμβάνονται, που θα απεικονίζουμε το μήνυμα επιτυχίας ή αποτυχίας, εισάγουμε συναρτησεις `usartcheck`.

Συνολικά στον κώδικα χρησιμοποιούμε 4 διαφορετικές συναρτήσεις `usartcheck` (`usartcheck1` , `usartcheck2`, `usartcheck3`, `usartcheck4`). Οι συναρτήσεις αυτές εκτελούν ουσιαστικά το ίδιο πράγμα, αλλά σε κάθε συνάρτηση εισάγονται παραλαγές αναγκαίες ανάλογα με το μήνυμα που θέλουμε να στείλουμε:

- Στέλνουμε ένα μήνυμα (εντολή) μέσω USART χρησιμοποιώντας την "`usart_transmit`"
- Περιμένουμε μια απάντηση και τη διαβάζουμε χαρακτήρα προς χαρακτήρα.
- Προσθέτουμε μια καθυστέρηση
- Ελέγχουμε εάν η ληφθείσα απάντηση είναι "Success" χρησιμοποιώντας το `strstr`.
- Η συνάρτηση επιστρέφει 1 εάν βρεθεί "Success" στην απάντηση, διαφορετικά επιστρέφει 0.

Αυτές οι λειτουργίες έχουν σχεδιαστεί για να χρησιμοποιούνται στον κύριο κώδικα για αποστολή συγκεκριμένων εντολών μέσω USART, λήψη και επεξεργασία των απαντήσεων και λήψη κατάλληλων ενεργειών με βάση την επιτυχία ή την αποτυχία των εντολών.

Στο δεύτερο μέρος του κώδικα:

A. Διαβάζουμε μια μέτρηση από τον αισθητήρα θερμοκρασίας DS18B20, προσθέτουμε στην τιμή του αισθητήρα την τιμή 12 ώστε το αποτέλεσμα μας να είναι κοντά στο 36 για να προσομοιάζει πραγματική θερμοκρασία ασθενούς.
B. Διαβάζουμε μια μέτρηση από το ποτενσιόμετρο και την μετατρέπουμε σε κλίμακα 0-20 cm H20 για να συμπεριφέρεται σαν αισθητήρας κεντρικής φλεβικής πίεσης.

Επιπλέον εισάγουμε την συνάρτηση `status` η οποία ανανεώνει το status ως εξής:

1. Αν πατηθεί στο πληκτρολόγιο το πλήκτρο 0 (που αντιστοιχεί στο τελευταίο ψηφίο της ομάδας μας) τότε το status γίνεται NURSE CALL. Αν πατηθεί στη συνέχεια η δίεση # τότε το status να γίνεται OK εκτός αν συμβαίνει κάτι από τα 2 η 3.
2. Αν η πίεση είναι πάνω από 12 ή κάτω από 4 τότε το status να γίνεται CHECK PRESSURE.
3. Αν η θερμοκρασία είναι κάτω από 34 ή πάνω από 37 τότε το status να γίνεται CHECK TEMP.
4. Αν δεν έχει συμβεί κανένα από τα παραπάνω τότε το status γίνεται OK.

Εμφανίζουμε τις τιμές της θερμοκρασίας και του αισθητήρα κεντρικής φλεβικής πίεσης στην 1η γραμμή της LCD και το status στην 2η γραμμή.

Στο τρίτο μέρος του κώδικα, ενσωματώνουμε τις παραπάνω πληροφορίες (θερμοκρασία, πίεση, status) στο payload.
Στην συνέχεια στέλνουμε στην UART κατάλληλες εντολές για να στείλουμε το payload στον server.
Αυτές είναι η εντολή payload για την οποία αντίστοιχα να εμφανίζουμε μηνύματα 3.Success και 3.Fail

και η εντολή transmit. Για την transmit δεν στέλνεται το ESP Success η Fail αλλά στέλνεται κατευθείαν την απάντηση του Server.

Στον κώδικα μας χρησιμοποιούμε όλες τις συναρτήσεις που έχουμε δημιουργήσει στις προηγούμενες εργασίες μας, για την οθόνη, για το πληκτρολόγιο, για την αρχικοποίηση του one wire protocol κλπ. Στις ήδη υπάρχουσες συναρτήσεις έχουμε εισάγει και κάποιες καινούργιες στον κύριο κώδικα μας.

```
void LCD_string(const char *str) {
    while (*str != '\0') {
        LCD_data(*str);
        _delay_ms(10);
        str++;
    }
}
```

Η συνάρτηση `LCD_string()` διευκολύνει την εμφάνιση μιας συμβολοσειράς στην LCD οθόνη μας, στέλνοντας κάθε χαρακτήρα της συμβολοσειράς στην οθόνη LCD έναν κάθε φορά.

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

unsigned char duty[] = {0x05, 0x1A, 0x2E, 0x43, 0x57, 0x6C, 0x80,
0x94, 0xA7, 0xBD, 0xD2, 0xE6, 0xFB};

uint16_t get_pressure() {

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

    unsigned char DC_VALUE = 6;
    OCR1AL = duty[DC_VALUE];

    //Set Vref = 5V
    ADMUX = (1<<REFS0) | (1<<ADLAR) ;
```

```
//Enable ADC & set ADC prescaler to 128
ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) |(1 << ADPS0);

unsigned char input, output;

input = PINC;
if ((input == 0b00011111) && (DC_VALUE != 12)) {
    //_delay_ms(100);
    while (PINC == 0b00011111) {
    }
    DC_VALUE++;
    //continue;

} else if ((input == 0b00101111) && (DC_VALUE != 0)) {
    //_delay_ms(100);
    while (PINC == 0b00101111) {
    }
    DC_VALUE--;

    //continue;
}


OCR1AL = duty[DC_VALUE];

_delay_ms(100);
ADCSRA |= (1<<ADSC);

output = ADCSRA;
output &= 0b01000000;
while(output){
    output = ADCSRA;
    output &= 0b01000000;
}

uint16_t pressure = ADC;
```

```
    return pressure;
}
```

Η συνάρτηση `get_pressure()` ενεργοποιεί τον ADC, ρυθμίζει τον prescaler ADC στο 128 για ακριβή μετατροπή ADC και διαβάζει το ποτενσιόμετρο. Εκκινεί μια μετατροπή ADC και περιμένει την ολοκλήρωσή της, ανακτά το αποτέλεσμα ADC στη μεταβλητή πίεσης και επιστρέφει την τιμή πίεσης.

```
int16_t get_temperature() {
    uint8_t temperatureHigh, temperatureLow;
    if (one_wire_reset()) {
        one_wire_transmit_byte(0xCC); // Skip ROM Command
        one_wire_transmit_byte(0x44); // Start temperature conversion

        // Wait for conversion to complete
        while (!one_wire_receive_bit()) {
        }

        if (one_wire_reset()) {
            one_wire_transmit_byte(0xCC); // Skip ROM Command
            one_wire_transmit_byte(0xBE); // Read Scratchpad Command
            temperatureLow = one_wire_receive_byte();
            temperatureHigh = one_wire_receive_byte();

            // Convert the temperature
            int16_t temperatureFinal = (temperatureHigh << 8) |
temperatureLow;
            return temperatureFinal;
        }

    }
    return 0x8000;

}
```

Συνοπτικά, η συνάρτηση `get_temperature()` διαβάζει μια ένδειξη θερμοκρασίας από τον αισθητήρα DS18B20 χρησιμοποιώντας το πρωτόκολλο One-Wire. Ξεκινά μια μετατροπή θερμοκρασίας, περιμένει να ολοκληρωθεί, διαβάζει τα δεδομένα θερμοκρασίας και επιστρέφει τη θερμοκρασία ως ακέραιος αριθμός 16-bit.