# 8η Εργαστηριακή Άσκηση

## Εργαστήριο Μικροϋπολογιστών


Πίκουλας Κωνσταντίνος 03120112
Στάμος Ανδρέας 03120018

### Βοηθητικά αρχεία

Παραθέτουμε τον κώδικα του αρχείου «usart.h»/«usart.c»/«esp8266.h»/«esp8266.c».

Ουσιαστικά έχουμε ρυθμίσει κατάλληλα την ασύγχρονη επικοινωνία UART για να επικοινωνεί ο AVR με τον ESP.

Επίσης το αρχείο «esp8266.c» περιέχει τον κώδικα για τα commands που χρειαζόμαστε στα πλαίσια της εργαστηριακής άσκησης.

```c
// usart.c

#include<stdint.h>
#include<avr/io.h>

#include "usart.h"

#define BUFFER_END '\n'
```

```c
void usart_init(uint16_t ubrr) {
    UCSR0A = 0;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UBRR0H = (uint8_t) (ubrr >> 8);
    UBRR0L = (uint8_t) (ubrr & 0xff);
    UCSR0C = 3 << UCSZ00; //1<<UCSZ01 | 1<<UCSZ00; //8-bit FIXME (maybe wrong)
    //no parity
    //1 stop bit
    //async mode
    //no interrupts
}

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

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

void usart_transmit_buffer(uint8_t *buf, int length) {
    while (length-- > 0) usart_transmit(*buf++);
}

int usart_receive_buffer(uint8_t *buf, int length) {
    int bread = 0;
    while (bread<length && ((*buf++ = usart_receive()) != BUFFER_END)) ++bread;
    return bread;
}
```

```c
// usart.h

#ifndef __USART_H
#define __USART_H

#include<stdint.h>

void usart_init(uint16_t ubrr);
void usart_transmit(uint8_t data);
uint8_t usart_receive(void);
```

```
void usart_transmit_buffer(uint8_t *buf, int length);
int usart_receive_buffer(uint8_t *buf, int length);

#endif
```

```
// esp8266.c

#include <string.h>
#include "usart.h"
#include "esp8266.h"

void esp8266_init(void) {
    usart_init(103); //baud rate 9600
}

int8_t esp8266_command(uint8_t command_type, uint8_t *arg, int arg_length) {
    //returns 0 if success, 1 if fail, -1 if other error
    static uint8_t connect[] = "ESP:connect";
    static uint8_t url[] = "ESP:url:";
    static uint8_t payload[] = "ESP:payload:";
    static uint8_t transmit[] = "ESP:transmit";

    static uint8_t success[] = "\"Success\"\n";
    static uint8_t fail[] = "\"Failure\"\n";

    uint8_t buf[50];
    int buf_lim;

    switch (command_type) {
        case CMD_CONNECT:
            usart_transmit_buffer(connect, sizeof(connect)-1);
            break;
        case CMD_URL:
            usart_transmit_buffer(url, sizeof(url)-1);
            usart_transmit_buffer(arg, arg_length);
            break;
        case CMD_PAYLOAD:
            usart_transmit_buffer(payload, sizeof(payload)-1);
            usart_transmit_buffer(arg, arg_length);
            break;
        case CMD_TRANSMIT:
            usart_transmit_buffer(transmit, sizeof(transmit)-1);
            break;
        default: return -1; break;
```

```
    }
    usart_transmit('\n');

    buf_lim = usart_receive_buffer(buf, sizeof(buf));

    switch (command_type) {
        case CMD_CONNECT:
        case CMD_URL:
        case CMD_PAYLOAD:
            if (!strncmp(buf, success, buf_lim)) return 0;
            else if (!strncmp(buf, fail, buf_lim)) return 1;
            break;
        case CMD_TRANSMIT:
            {
                int ret;
                for (ret=0; ret<buf_lim && ret<arg_length; ++ret) arg[ret] =
buf[ret];
                return ret;
                break;
            }
        default:
            return -1;
            break;
    }
    return -1;
}
```

```
// esp8266.h

#ifndef __ESP8266_H
#define __ESP8266_H

#include<stdint.h>

#define CMD_CONNECT 0
#define CMD_URL     1
#define CMD_PAYLOAD 2
#define CMD_TRANSMIT    3


void esp8266_init(void);
int8_t esp8266_command(uint8_t command_type, uint8_t *arg, int arg_length);
#endif
```

## Ζήτημα 8.1

Στέλνουμε κατάλληλες εντολές στον ESP μέσω του AVR ώστε να γίνει η σύνδεση στο διαδίκτυο και να δεχτεί το URL ο ESP. Στην LCD τυπώνουμε αντίστοιχα τα μηνύματα επιτυχίας «1.Success» και «2.Success».

Το κύριο σκέλος του κώδικα φαίνεται παρακάτω:

```
#include<util/delay.h>
#include<avr/io.h>
#include "../common/esp8266.h"
#include "../common/lcd.h"
#include "../common/usart.h"

#define DELAY 1500

int main() {
    uint8_t url[] = "\"http://192.168.1.250:5000/data\"";

    uint8_t success1_msg[] = "1.Success";
    uint8_t fail1_msg[] = "1.Fail";
    uint8_t success2_msg[] = "2.Success";
    uint8_t fail2_msg[] = "2.Fail";

    int8_t ret;

    esp8266_init();
    lcd_init();

    ret = esp8266_command(CMD_CONNECT, 0, 0);
    lcd_clear_display();
    switch (ret) {
        case 0:
            lcd_data_buf(success1_msg, sizeof(success1_msg)-1);
            break;
        case 1:
            lcd_data_buf(fail1_msg, sizeof(fail1_msg)-1);
            goto end;
            break;
        default:
            goto end;
            break;
    }

    _delay_ms(DELAY);
```

```
    ret = esp8266_command(CMD_URL, url, sizeof(url)-1);
    lcd_clear_display();
    switch (ret) {
        case 0:
            lcd_data_buf(success2_msg, sizeof(success2_msg)-1);
            break;
        case 1:
            lcd_data_buf(fail2_msg, sizeof(fail2_msg)-1);
            goto end;
            break;
        default:
            goto end;
            break;
    }
end:
    while(1);
}
```

## Ζήτημα 8.2/8.3

Εφόσον το ζήτημα 8.3 περιέχει τον κώδικα του ζητήματος 8.2, παραθέτουμε τον κώδικα της άσκησης 8.3:

Το τελευταίο ψηφίο της ομάδας μας είναι το ‘7’.

Πατώντας ‘#’ στο πλήκτρο το STATUS του ασθενή γίνεται OK εκτός αν δεν πληρούνται οι περιορισμοί για την θερμοκρασία και την πίεση. Την πίεση την προσομοιάζουμε ρυθμίζοντας κατάλληλα το ποτενσιόμετρο POT0 και μετατρέποντας την κλίμακα σε 0-20 cm H20.

Ορίσαμε ένα offset για την θερμοκρασία ίσο με ‘11’. Προστίθεται στην θερμοκρασία του επιστρέφει το θερμόμετρο προκειμένου να προσομοιάζουμε πραγματικές συνθήκες.

Ο κώδικας εκτελεί τα κατάλληλα delays μεταξύ κάθε φάσης του προγράμματος προκειμένου να φαίνονται στο LCD Screen τα δεδομένα.

Στέλνοντας το payload στον ESP βλέπουμε να επιστρέφει «200 OK» και το τυπώνουμε στο LCD Screen.

```
#include<stdio.h>
#include<util/delay.h>
#include<avr/io.h>
#include "../common/esp8266.h"
#include "../common/lcd.h"
#include "../common/usart.h"
```

```c
#include "../common/thermometer.h"
#include "../common/adc.h"
#include "../common/keypad.h"

#define DELAY 1500
#define NURSE_DIGIT '7'
#define NURSE_CANCEL '#'

#define TEMP_OFFSET 11

int main() {
    static uint8_t url[] = "\"http://192.168.1.250:5000/data\"";

    static uint8_t success1_msg[] = "1.Success";
    static uint8_t fail1_msg[] = "1.Fail";
    static uint8_t success2_msg[] = "2.Success";
    static uint8_t fail2_msg[] = "2.Fail";
    static uint8_t success3_msg[] = "3.Success";
    static uint8_t fail3_msg[] = "3.Fail";

    int8_t ret;

    esp8266_init();
    lcd_init();
    keypad_init();

    ret = esp8266_command(CMD_CONNECT, 0, 0);
    lcd_clear_display();
    switch (ret) {
        case 0:
            lcd_data_buf(success1_msg, sizeof(success1_msg)-1);
            break;
        case 1:
            lcd_data_buf(fail1_msg, sizeof(fail1_msg)-1);
            goto end;
            break;
        default:
            goto end;
            break;
    }

    _delay_ms(DELAY);
    ret = esp8266_command(CMD_URL, url, sizeof(url)-1);
    lcd_clear_display();
    switch (ret) {
```

```
        case 0:
            lcd_data_buf(success2_msg, sizeof(success2_msg)-1);
            break;
        case 1:
            lcd_data_buf(fail2_msg, sizeof(fail2_msg)-1);
            goto end;
            break;
        default:
            goto end;
            break;
    }

    adc_init();

    static uint8_t ok_msg[] = "OK";
    static uint8_t nurse_msg[] = "NURSECALL";
    static uint8_t checkpres_msg[] = "CHECKPRESSURE";
    static uint8_t checktemp_msg[] = "CHECKTEMP";

    uint8_t nurse = 0;
    while (1) {
        float t = get_temperature_f();
        t += TEMP_OFFSET;
        float p = (((float) adc_measure()) * 20.0 / 1024.0);
        switch (keypad_to_ascii()) {
            case NURSE_DIGIT: nurse = 1; break;
            case NURSE_CANCEL: nurse = 0; break;
        }

        char *status = ok_msg;
        int status_lim = sizeof(ok_msg)-1;
        if (nurse) {
            status = nurse_msg;
            status_lim = sizeof(nurse_msg)-1;
        } else {
            if (p > 12 || p < 4) {
                status = checkpres_msg;
                status_lim = sizeof(checkpres_msg)-1;
            }
            if (t < 34 || t > 37) {
                status = checktemp_msg;
                status_lim = sizeof(checktemp_msg)-1;
            }
        }
```

```
        char buf[300];
        int buf_lim;

        buf_lim = snprintf(buf, sizeof(buf), "%.1f %.1f", t, p);
        lcd_clear_display();
        lcd_data_buf(buf, buf_lim);
        lcd_goto_line2();
        lcd_data_buf(status, status_lim);
        _delay_ms(500);

        buf_lim = snprintf(buf, sizeof(buf), "[{\"name\":
\"temperature\",\"value\": \"%.1f\"},{\"name\": \"pressure\",\"value\""
                ": \"%.1f\"},{\"name\": \"team\",\"value\": \"17\"},{\"name\":
\"status\",\"value\": \"%s\"}]", t, p, status);

        ret = esp8266_command(CMD_PAYLOAD, buf, buf_lim);
        lcd_clear_display();
        switch (ret) {
            case 0:
                lcd_data_buf(success3_msg, sizeof(success3_msg)-1);
                break;
            case 1:
                lcd_data_buf(fail3_msg, sizeof(fail3_msg)-1);
                goto end;
                break;
            default:
                goto end;
                break;
        }
        _delay_ms(500);

        buf[0] = '4';
        buf[1] = '.';
        buf_lim = esp8266_command(CMD_TRANSMIT, buf+2, sizeof(buf)-2);
        buf_lim += 2;
        lcd_clear_display();
        if (buf_lim<0) goto end;
        lcd_data_buf(buf, buf_lim);
        _delay_ms(500);


    }
end:
    while(1);
}
```

## Βοηθητικά αρχεία

Παραθέτουμε επίσης βοηθητικά αρχεία που είχαμε χρησιμοποιήσει από προηγούμενες εργαστηριακές ασκήσεις.

```c
// adc.c

#include<stdint.h>
#include<avr/io.h>
#include "adc.h"

void adc_init() {
    ADMUX = 1 << REFS0;
    ADCSRA = 1 << ADEN | 1 << ADPS0 | 1 << ADPS1 | 1 << ADPS2;
    ADCSRB = 0;
    DIDR0 = 1 << ADC0D;
}

uint16_t adc_measure() {
    ADCSRA |= (1 << ADSC);
    while(ADCSRA & 1<<ADSC);
    return ADC;
}
```

```c
// adc.h

#ifndef __ADC_H
#define __ADC_H

void adc_init();
uint16_t adc_measure();

#endif
```

```c
// keypad.c

#include<util/delay.h>
#include "twi_pca9555.h"
#include<avr/io.h>

static uint8_t scan_row(uint8_t row) {
    //io1_{row} pull down to zero. (rows measured from bottom to top)
    PCA9555_0_write(REG_OUTPUT_1, (uint8_t) ~(1U<<row));
    uint8_t pressed = PCA9555_0_read(REG_INPUT_1);
```

```
    return (uint8_t) (~pressed >> 4) & 0x0f;
}

static uint16_t scan_keypad(void) {
    //MSB 4-bits are 1st line, 2nd MSB 4-bits are 2nd line, ...
    return (uint16_t) ((scan_row(3) << 12) | (scan_row(2) << 8) | (scan_row(1)
<< 4) | scan_row(0));
}

static uint16_t scan_keypad_rising_edge(void) {
    static uint16_t keypad_before = 0; //static variable preserved across all
function calls
    uint16_t keypad_1, keypad_2;
    keypad_1 = scan_keypad();
    _delay_ms(30);
    keypad_2 = scan_keypad();
    uint16_t ret = keypad_1 & keypad_2 & ~keypad_before; //should be pressed
both times now, and not be pressed in previous call
    keypad_before = keypad_1 & keypad_2; //storing currently pressed keys
    return ret;
}

static char uint16_to_ascii(uint16_t keypad) {
    //unset all bits except the MSB 1, to ensure that it will work even if two
buttons are pressed
    uint16_t msb_mask = 0x8000;
    while (msb_mask > 0 && ~keypad & msb_mask) msb_mask >>= 1;
    keypad &= msb_mask;

    switch (keypad) {
        case 1U<<12: return '1'; break;
        case 1U<<13: return '2'; break;
        case 1U<<14: return '3'; break;
        case 1U<<15: return 'A'; break;

        case 1U<<8:  return '4'; break;
        case 1U<<9:  return '5'; break;
        case 1U<<10: return '6'; break;
        case 1U<<11: return 'B'; break;

        case 1U<<4:  return '7'; break;
        case 1U<<5:  return '8'; break;
        case 1U<<6:  return '9'; break;
        case 1U<<7:  return 'C'; break;
```

```
        case 1U<<0:  return '*'; break;
        case 1U<<1:  return '0'; break;
        case 1U<<2:  return '#'; break;
        case 1U<<3:  return 'D'; break;
    }
    return 0;
}

char keypad_to_ascii(void) {
    return uint16_to_ascii(scan_keypad_rising_edge());
}

char keypad_to_ascii_pressed(void) {
    return uint16_to_ascii(scan_keypad());
}

void keypad_init(void) {
    twi_init();
    //pca9555 io1[0] as output, io1[4:7] as input;
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
}
```

```
// keypad.h

#ifndef __KEYPAD_H
#define __KEYPAD_H

char keypad_to_ascii_pressed(void);
char keypad_to_ascii(void);
void keypad_init(void);
#endif
```

```
// thermometer.c

#include "onewire.h"
#include "thermometer.h"

int16_t get_temperature(void) {
    if (!one_wire_reset()) return 0x8000L;
    one_wire_transmit_byte(0xCC);
    one_wire_transmit_byte(0x44);
    while (!one_wire_receive_bit());
    if (!one_wire_reset()) return 0x8000L;
```

```
    one_wire_transmit_byte(0xCC);
    one_wire_transmit_byte(0xBE);

    int16_t ret;
    ret = one_wire_receive_byte(); //LSB byte
    ret |= one_wire_receive_byte() << 8; //MSB byte

    return ret;
}

float get_temperature_f(void) {
    int16_t t = get_temperature();

        uint8_t is_neg = t<0;
        if (t<0) t=-t;

        float t_fl = ((int16_t) (t >> 4)) + ((float)(t & 0xf))/16.0;
        if (is_neg) t_fl = -t_fl;

    return t_fl;
}
```

```
// thermometer.h

#ifndef __THERMOMETER_H
#define __THERMOMETER_H

#include<stdint.h>

int16_t get_temperature(void);
float get_temperature_f(void);

#endif
```

```
// onewire.c

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

#include "onewire.h"

#define SETOUTPUT() do { DDRD |= (uint8_t) 1U<<4; } while (0)
```

```c
#define OUTPUT_0() do { PORTD &= (uint8_t) ~(1U<<4); } while (0)
#define OUTPUT_1() do { PORTD |= (uint8_t) 1U<<4; } while (0)
#define OUTPUT(x) OUTPUT_##x()

#define SETINPUT() do { DDRD &= (uint8_t) ~(1U<<4); PORTD &= (uint8_t)
~(1U<<4); } while (0)
#define READ() ((PIND >> 4) & 1U)

uint8_t one_wire_reset(void) {
    SETOUTPUT();
    OUTPUT(0);
    _delay_us(480);
    SETINPUT();
    _delay_us(100);
    uint8_t ret = (READ() == 0);
    _delay_us(380);
    return ret;
}

uint8_t one_wire_receive_bit(void) {
    SETOUTPUT();
    OUTPUT(0);
    _delay_us(2);
    SETINPUT();
    _delay_us(10);
    uint8_t ret = READ();
    _delay_us(49);
    return ret;
}

void one_wire_transmit_bit(uint8_t datum) {
    SETOUTPUT();
    OUTPUT(0);
    _delay_us(2);
    if (datum & 1) OUTPUT(1);
    else OUTPUT(0);
    _delay_us(58);
    SETINPUT();
    _delay_us(1);
}

uint8_t one_wire_receive_byte(void) {
    uint8_t ret = 0;
    for (int8_t i=0; i<8; ++i) ret |= one_wire_receive_bit() << i;
    return ret;
```

```
}

void one_wire_transmit_byte(uint8_t data) {
    for (int8_t i=0; i<8; ++i) {
        one_wire_transmit_bit(data & 1);
        data >>= 1;
    }
}
```

```
// onewire.h

#ifndef __ONEWIRE_H
#define __ONEWIRE_H

#include<stdint.h>

uint8_t one_wire_reset(void);
uint8_t one_wire_receive_bit(void);
void one_wire_transmit_bit(uint8_t);
uint8_t one_wire_receive_byte(void);
void one_wire_transmit_byte(uint8_t);

#endif
```

```
// twi_pca9555.c

#include<avr/io.h>
#include<util/twi.h>
#include<stdint.h>

#include "twi_pca9555.h"

#define PCA9555_0_ADDRESS 0x40

#define SCL_CLOCK 100000L
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK-16)/2)

#define TW0_STATUS (TWSR0 & 0xF8)


void twi_init(void) {
    TWSR0 = 0; //prescaler = 1
    TWBR0 = TWBR0_VALUE;
}
```

```
unsigned char twi_start(uint8_t address) {
    //7 msb of address are address and lsb is set if R and not set if W

    TWCR0 = 1<<TWSTA | 1<<TWINT | 1<<TWEN; //START
    while (!(TWCR0 & (1<<TWINT))); //wait till START transmitted
    if (TW0_STATUS != TW_START && TW0_STATUS != TW_REP_START) return 1;
//failed

    TWDR0 = address; //SLA_W or SLA_R
    TWCR0 = 1<<TWINT | 1<<TWEN;

    while (!(TWCR0 & (1<<TWINT))); //wait till SLA transmitted

    //if SLA_R
    if (address & 0x01) {
        if (TW0_STATUS != TW_MR_SLA_ACK) return 1; //failed
    } else {
    //if SLA_W
        if (TW0_STATUS != TW_MT_SLA_ACK) return 1; //failed
    }

    return 0;
}

void twi_stop(void) {
    TWCR0 = 1<<TWSTO | 1<<TWINT | 1<<TWEN; //STOP
    while (!(TWCR0 & (1<<TWSTO))); //wait till STOP transmitted
}

unsigned char twi_write(uint8_t data) {
    TWDR0 = data;
    TWCR0 = 1<<TWINT | 1<<TWEN;
    while (!(TWCR0 & (1<<TWINT))); //wait till transmitted
    if (TW0_STATUS != TW_MR_DATA_ACK) return 1;
    return 0;
}

unsigned char twi_readAck(void) {
    TWCR0 = 1<<TWINT | 1<<TWEA | 1<<TWEN;
    while (!(TWCR0 & (1<<TWINT))); //wait till received
    return TWDR0;
}

unsigned char twi_readNak(void) {
```

```
    TWCR0 = 1<<TWINT | 1<<TWEN;
    while (!(TWCR0 & (1<<TWINT))); //wait till received
    return TWDR0;
}

void twi_start_wait(uint8_t address) {
    while (twi_start(address));
}

void PCA9555_0_write(uint8_t reg, uint8_t value) {
    twi_start_wait(PCA9555_0_ADDRESS | TW_WRITE);
    twi_write(reg); //what if PCA9555 NACKs?
    twi_write(value);
    twi_stop();
}

uint8_t PCA9555_0_read(uint8_t reg) {
    uint8_t ret_val;
    twi_start_wait(PCA9555_0_ADDRESS | TW_WRITE);
    twi_write(reg); //what if PCA9555 NACKs?
    twi_start(PCA9555_0_ADDRESS | TW_READ); //what if fails? should we wait
instead?
    ret_val = twi_readNak();
    twi_stop();
    return ret_val;
}
```

```
// twi_pca9555.h

#ifndef __TWI_PCA9555_H
#define __TWI_PCA9555_H

#include<stdint.h>

#define REG_INPUT_0     0
#define REG_INPUT_1     1
#define REG_OUTPUT_0        2
#define REG_OUTPUT_1        3
#define REG_POLARITY_INV_0  4
#define REG_POLARITY_INV_1  5
#define REG_CONFIGURATION_0 6
#define REG_CONFIGURATION_1 7

void twi_init(void);
```

```
unsigned char twi_start(uint8_t address);
void twi_stop(void);
unsigned char twi_write(uint8_t data);
unsigned char twi_readAck(void);
unsigned char twi_readNak(void);
void twi_start_wait(uint8_t address);

void PCA9555_0_write(uint8_t reg, uint8_t value);
uint8_t PCA9555_0_read(uint8_t reg);

#endif
```