Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

ΑΘΗΝΑ 16 Δεκεμβρίου 2022

8η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"
Συνδυαστική/Επαναληπτική άσκηση – Εφαρμογή Internet of Things (στο ntuAboard)

Αναφορά 8ης Εργαστηριακής Άσκησης

Ραπτόπουλος Πέτρος (el19145)
Σαφός Κωνσταντίνος (el19172)

Ζήτημα

Παρακάτω φαίνεται ο κώδικας σε C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε C

```c
#define F_CPU 16000000UL
#include <math.h>
#include <util/delay.h>
#include <avr/io.h>
#include <string.h>
#include <stdio.h>

// --------------------------- UART -------------------------------------
/*
 Routine: usart_init
    Description:
        This routine initializes the
        usart as shown below.
------- INITIALIZATIONS -------
    Baud rate: 9600 (Fck= 8MH)
    Asynchronous mode
    Transmitter on
    Reciever on
    Communication parameters: 8 Data ,1 Stop, no Parity
--------------------------------
    parameters: ubrr to control the BAUD.
    return value: None.
 */

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

/*
Routine: usart_transmit
    Description:
        This routine sends a byte of data
        using usart.
    parameters:
    data: the byte to be transmitted
    return value: None.
 */
void usart_transmit(uint8_t data){
    while(!(UCSR0A&(1<<UDRE0)));
    UDR0=data;
}

/*
Routine: usart_receive
    Description:
        This routine receives a byte of data
        from usart.
    parameters: None.
    return value: the received byte
 */
uint8_t usart_receive(){
    while(!(UCSR0A&(1<<RXC0)));
    return UDR0;
}
```

```c
void usart_transmit_text(char* text) {
    int index = 0;
    do {
        usart_transmit((uint8_t)text[index]);
    } while(text[index++] != '\n');
}

uint8_t response[200]; // max length of message

void usart_receive_text() {
    for(int i = 0; i < 200; i++)
        response[i] = '\0';
    int index = 0;
    do {
        response[index] = usart_receive();
    } while(response[index++] != '\n');
    response[index] = '\0';
}

// ------------------- TEMPERATURE SENSOR --------------------------------

uint8_t one_wire_reset(void) {
    DDRD = 0b00010000; //PD4 output
    PORTD = PORTD & 0b11101111; //PD4 = 0;
    _delay_us(480); //480 usec reset pulse

    DDRD = DDRD & 0b11101111; //set PD4 as input
    PORTD = PORTD & 0b11101111; // disable pull-up
    _delay_us(100); //wait 100 usec for connected devices to transmit the presence pulse

    uint8_t state = PIND;
    _delay_us(380);

    if((state & 0x10) == 0x00)
        return 0x01;
    return 0x00;
    // returns 1 if PD4 = 0
    // if a connected device is detected(PD4=0) return 1 else return 0
}


uint8_t one_wire_receive_bit(void) {
    DDRD = 0b00010000; //PD4 output
    PORTD = PORTD & 0b11101111; //PD4 = 0;
    _delay_us(2); //time slot 2 usec

    DDRD = 0b11101111; //set PD4 as input
    PORTD = PORTD & 0b11101111; // disable pull-up
    _delay_us(10);

    uint8_t state = PIND;
    _delay_us(49); //delay 49 usec to meet the standards

    if((state & 0x10) == 0x10)
        return 0x01;
    return 0x00; // returns 1 if PD4 is 1
}

void one_wire_transmit_bit(uint8_t data) {
    DDRD = 0b00010000; //PD4 output
    PORTD = PORTD & 0b11101111; //PD4 = 0;
    _delay_us(2);

    PORTD = (PORTD & 0xEF) | ((data << 4) & 0x10);
    // alter the PD4 of current PORTD's state and make it 0.
    // then bitwise or with data
    // we want to change only the PD4 bit
    _delay_us(58); //wait 58 usec for connected device to sample the line

    DDRD = DDRD & 0b11101111; //set PD4 as input
    PORTD = PORTD & 0b11101111; // disable pull-up
    _delay_us(1); //recovery time 1 usec
}
```

```c
uint8_t one_wire_receive_byte(void) { //starts from LSB
    uint8_t data = 0;
    for (int i = 0; i < 8; i++) {
        uint8_t x = one_wire_receive_bit(); //x[0] holds the received bit
        data = data >> 1;
        if (x == 0x01)
            data = data | 0x80;
    }
    return data;
}

void one_wire_transmit_byte(uint8_t data) { //starts from LSB
    for (int i = 0; i < 8; i++) {
        one_wire_transmit_bit(data);
        data = data >> 1;
    }
}

int read_temp(void) { // the value returned is the temperature
                      // with 0.0625 degrees decision (DS18?20 sensor)
    if (one_wire_reset() == 0x01) {
        one_wire_transmit_byte(0xCC); //0xCC skip device choice

        one_wire_transmit_byte(0x44); //0x44 start temperature reading

        while (one_wire_receive_bit() == 0x00); //wait for reading completion

        one_wire_reset();             // new reset of the device
        one_wire_transmit_byte(0xCC); //0xCC skip device choice

        one_wire_transmit_byte(0xBE); //0xBE read temperature

        uint8_t temperature = one_wire_receive_byte();
        uint8_t sign = one_wire_receive_byte();
        return (((uint16_t)sign) << 8) + (uint16_t)temperature;
        // combine the two 8bits numbers to one comprised of 16bits
    } else return 0x8000;
}

//-----------------------------LCD SCREEN---------------------------------

// send one byte divided into 2 (4 bit) parts

void write_2_nibbles(char data) {
    char pinState = PIND; // read 4 LSB and resend them
    // in order not to alter any previous state
    PORTD = (pinState & 0x0F) | (data & 0xF0) | (1 << PD3); //  4MSB
    PORTD &= (0xFF) & (0 << PD3); // set PD3 to zero, lcd enable pulse
    PORTD = (pinState & 0x0F) | ((data << 4) & 0xF0) | (1 << PD3); // send 4LSB
    PORTD &= (0xFF) & (0 << PD3); // set PD3 to zero, lcd enable pulse
}
// send one byte of data to the lcd display

void lcd_data(char data) {
    PORTD |= (1 << PD2); // select data register
    write_2_nibbles(data);
    _delay_us(100);
}
// send one byte of instuction to the lcd display

void lcd_command(char data) {
    PORTD &= (0xFF) & (0 << PD2); // select command register
    write_2_nibbles(data);
    _delay_us(100);
}
```

```
void lcd_text(char* text, int line) {
    if (line != 2) {
        lcd_command(0x01); // clear display
        _delay_us(5000);
    } else if (line == 2) {
        lcd_command(0b11000000); //write to second line (DDRAM Address 0x40)
        _delay_us(5000);
        lcd_command(0b11000000); //write to second line (DDRAM Address 0x40)
        _delay_us(5000);
    }
    int index = 0;
    while(text[index] != '\n') {
        lcd_data(text[index++]);
        _delay_ms(100);
    }
}

void lcd_init() {
    _delay_ms(40); // lcd init procedure
    PORTD = 0x30 | (1 << PD3); // 8 bit mode
    PORTD &= (0xFF) & (0 << PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x30 | (1 << PD3); // 8 bit mode
    PORTD &= (0xFF) & (0 << PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x20 | (1 << PD3); // change to 4 bit mode
    PORTD &= (0xFF) & (0 << PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    lcd_command(0x28); // select character size 5x8 dots and two line display
    lcd_command(0x0c); // enable lcd, hide cursor
    lcd_command(0x01); // clear display
    _delay_us(5000);
    lcd_command(0x06); // enable auto increment of address, disable shift of the display
    //lcd_command(0x07); // enable auto increment of address, enable shift of the display
    //lcd_command(0x1C); // enable auto increment of address, enable shift of the display
    //lcd_command(0x18); // enable auto increment of address, enable shift of the display
}

//------------------------------Temperature----------------------------------

int convertTemp(uint16_t temperature, int* intTemperature, int* decTemperature) {
    if (temperature == 0x8000) { //NO DEVICE
            lcd_text("NO DEVICE\n", 1);
            return -1; // error value
    } else {
        int sign;
        if ((temperature & 0x8000) == 0x1000) { //negative
            sign = -1; // negative
            temperature = ~temperature; // 2's complement
            temperature += 1;
        } else sign = 1; // positive
        // now variable temperature holds the absolute value
        // of the temperature read by the sensor
        int result = 0;
        uint8_t intTemp = (temperature >> 4); // integer part of temperature
        char hund = (intTemp / 100); // hundreds part of temperature
        result += hund*100;

        char tens = ((intTemp % 100) / 10); // tens part of temperature
        result += tens*10;

        char ones = (intTemp % 10);      //ones part of temperature
        result += ones;

        char firstDecimal = ((int) (temperature*0.0625 * 10) % 10); // 1st decimal

        *intTemperature = result*sign;
        *decTemperature = firstDecimal;
        return 0; // no error found
    }
}
```

```
int temperatureOffset = -1;

void send_packet(uint16_t temperature, float pressure, char* keyboardStatus) {

    // ---------------------- Make Conversion --------------------

    int intTemp, decTemp;
    convertTemp(temperature, &intTemp, &decTemp);

    int intPress = (int)pressure;
    int decPress = (int)(pressure*10)%10;

    // ---------- Find and Apply Temperature Offset ----------------
    if (temperatureOffset == -1) temperatureOffset = 36 - intTemp;
    intTemp += temperatureOffset;
    // temperatureOffset is defined at first temperature measurement
    // is first measurement reliable? should we take the average of
    // the measurements?

    // -------------------- Find Status ---------------------------
    char* status = keyboardStatus;
    if (pressure > 12 || pressure < 4) status = "CHECKPRESSURE";
    if (intTemp < 34 || (intTemp >= 37 && decTemp != 0)) status = "CHECKTEMP";

// --------- Display Temperature - Pressure -------------------
    char lcdText[100];
    sprintf(lcdText, "T: %d.%d, P: %d.%d\n", intTemp, decTemp, intPress, decPress);
    lcd_text(lcdText, 1);
    _delay_ms(2000);

    sprintf(lcdText, "ST: %s\n", status);
    lcd_text(lcdText, 2); // write at second line
    _delay_ms(2000);

    int failed = 0;
    do {
        _delay_ms(500);
        if (failed) lcd_text("1.FAILED\n", 1);
        failed = 1;
        usart_transmit_text("ESP:connect\n");
        usart_receive_text();
    } while(strcmp(response, "\"Success\"\n"));

    lcd_text("1.SUCCESS\n", 1);
    _delay_ms(1000);

    failed = 0;
    do {
        _delay_ms(500);
        if (failed) lcd_text("2.FAILED\n", 1);
        failed = 1;
        usart_transmit_text("ESP:url:\"http://192.168.1.250:5000/data\"\n");
        usart_receive_text();
    } while(strcmp(response, "\"Success\"\n"));

    lcd_text("2.SUCCESS\n", 1);
    _delay_ms(500);

    failed = 0;
    do {
        // delay?
        if (failed) lcd_text("3.FAILED\n", 1);
        failed = 1;
        char payload[200];
        sprintf(payload,
"ESP:payload:[{\"name\":\"temperature\",\"value\":%d.%d},{\"name\":\"pressure\",\"value\":%d.%d},{\"name\"
:\"team\",\"value\":\"59\"},{\"name\":\"status\",\"value\":%s}]\n", intTemp, decTemp, intPress, decPress,
status);
        usart_transmit_text(payload);
        usart_receive_text();
    } while(strcmp(response, "\"Success\"\n"));
```

```
    lcd_text("3.SUCCESS\n", 1);
    _delay_ms(500);

    do {
        usart_transmit_text("ESP:transmit\n");
        usart_receive_text();
        sprintf(lcdText, "4.%s\n", response);
        lcd_text(lcdText, 1);
        _delay_ms(1000);
    } while(strcmp(response, "200 OK\n"));

}

// ---------------------------- POTENTIOMETER ----------------------------

float read_adc() {
    int adc = ADC;        // 0 < adc < 1023
    float CVP = adc*20.0/1024;    // 0 < CVP < 20
                    // CVP: central venous pressure in cm of water
    return CVP;
}

// ------------------------------ KEYBOARD ------------------------------
#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
#define TWI_READ 1 // reading from twi device
#define TWI_WRITE 0 // writing to twi device
#define SCL_CLOCK 100000L // twi clock in Hz
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2

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

//----------- Master Transmitter/Receiver -------------------
#define TW_START 0x08
#define TW_REP_START 0x10
//---------------- Master Transmitter ----------------------
#define TW_MT_SLA_ACK 0x18
#define TW_MT_SLA_NACK 0x20
#define TW_MT_DATA_ACK 0x28
//---------------- Master Receiver ----------------
#define TW_MR_SLA_ACK 0x40
#define TW_MR_SLA_NACK 0x48
#define TW_MR_DATA_NACK 0x58

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

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

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

```
// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address) {
    uint8_t twi_status;
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);        // send START condition
    while(!(TWCR0 & (1<<TWINT)));       // wait until transmission completed
    twi_status = TW_STATUS & 0xF8;  // check value of TWI Status Register.
    if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
    TWDR0 = address;    // send device address
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));   // wail until transmission completed and ACK/NACK has been received
    twi_status = TW_STATUS & 0xF8;  // check value of TWI Status Register.
    if ((twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK)) {
        return 1;
    }
    return 0;
}

// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address) {
    uint8_t twi_status;
    while (1) {
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);    // send START condition
    while(!(TWCR0 & (1<<TWINT)));  // wait until transmission completed
    twi_status = TW_STATUS & 0xF8; // check value of TWI Status Register.
    if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;
    TWDR0 = address;   // send device address
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));      // wail until transmission completed
    twi_status = TW_STATUS & 0xF8;     // check value of TWI Status Register.
    if ((twi_status == TW_MT_SLA_NACK ) || (twi_status ==TW_MR_DATA_NACK)) {
        //device busy, send stop condition to terminate write operation
        TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        while(TWCR0 & (1<<TWSTO)); // wait until stop condition is executed and bus released
        continue;
    }
    break;
    }
}

// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write(unsigned char data) {
    TWDR0 = data;       // send data to the previously addressed device
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));   // wait until transmission completed
    if ((TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
    return 0;
}

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

// Terminates the data transfer and releases the twi bus
void twi_stop(void) {
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);    // send stop condition
    while(TWCR0 & (1<<TWSTO));  // wait until stop condition is executed and bus released
}

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

```
uint8_t twi_readNak(void) {
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

uint8_t PCA9555_0_read(PCA9555_REGISTERS reg) {
    uint8_t ret_val;
    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
    ret_val = twi_readNak();
    twi_stop();
    return ret_val;
}

char scan_row(int row) {
    // row = 0, 1, 2, 3 for IO1_0, IO1_1, IO1_2, IO1_3
    // we set the row we are scanning to 0, while all the rest
    // are set to 1. if a key is pressed the correspondent column
    // would switch to 0. Otherwise it is 1.
    if(row == 0) PCA9555_0_write(REG_OUTPUT_1, 0x0E);
    else if(row == 1) PCA9555_0_write(REG_OUTPUT_1, 0x0D);
    else if(row == 2) PCA9555_0_write(REG_OUTPUT_1, 0x0B);
    else if(row == 3) PCA9555_0_write(REG_OUTPUT_1, 0x07);
    return PCA9555_0_read(REG_INPUT_1) & 0xF0; // read IO1[7:4]
}

void scan_keypad(char* keypad_state) {
// keypad_state holds the current state of all rows
    for(int i = 0; i <= 3; i++) // scan each row
        keypad_state[i] = scan_row(i);
}

int diff_bit(char c1, char c2) {
    // c1 holds the old(current) state
    // c2 holds the new state
    char diff = c1^c2;  // xor of c1, c2  we have 0 if
    char mask = 0x10;   // the ith bit of c1 and c2 are same
    for (int i = 0; i <= 3; ++i) {
        if(((diff & mask) != mask) && ((c2 & mask) == 0))
// we check c2 & mask == 0 in order to be notified only when the
// button is pressed. If unpressed ignore.
            return i; // different bit found
        mask = mask << 1;   // shift the mask to the left
    }
    return -1; // same
}

int same(char* array1, char* array2) { // array1, array2 have size 4
    for(int i = 0; i <= 3; ++i)
        if (array1[i]!=array2[i])
            return 0;
    return 1;
}

char keypad_state_current[4];   // hold the current keypad state
int scan_keypad_rising_edge() {
    char keypad_state_1[4];
    char keypad_state_2[4];
    // read two times the keypad, if the states are different then
    // bouncing is under way.
    do {
        scan_keypad(keypad_state_1);
        _delay_ms(15);
        scan_keypad(keypad_state_2);
    } while(same(keypad_state_1, keypad_state_2) == 0);

    int ret = -1;
    for(int i = 0; i <= 3; ++i) {
        int diff = diff_bit(keypad_state_current[i], keypad_state_1[i]);
```

```
// return which bit is different between current and previous keypad state
        keypad_state_current[i] = keypad_state_1[i]; // update current keypad_state
        if (diff != -1) ret = diff + 4*i;            // the place inside chars[] array
    }
    // 0: '*', 1: '0', 2: '#', 3: 'D', 4:'7', ...
    return ret; // same keypad states
}

char keypad_to_ascii() {
    int scan = scan_keypad_rising_edge();
    if(scan == -1) return 0; // no buttons are pressed
    char chars[16] = {'*', '0', '#', 'D', '7', '8', '9', 'C', '4', '5', '6', 'B', '1', '2', '3', 'A'};
    // the sequence of the above chars is the result of the way we scan the keypad
    return chars[scan];
}

// --------------------------- MAIN -----------------------------------------

int main(void) {

    // REFSn[1:0]=01 => select Vref=5V, MUXn[4:0]=0000 => select ADC0(POT1),
    // ADLAR=0 => Right adjust the ADC result
    ADMUX = 0b01000000;
    // ADEN=1 => ADC Enable, ADCS=0 => No Conversion,
    // ADIE=0 => disable adc interrupt, ADPS[2:0]=111 => fADC=16MHz/128=125KHz
    ADCSRA = 0b10000111;

    DDRD = 0xFF;
    DDRC = 0xFF;
    lcd_init(); // init lcd
    _delay_ms(2); // wait for lcd init

    /*
     We have UBRR = fosc/16BAUD - 1
     So in order to have BAUD = 9600 (fosc = 16MHz)
     we need UBRR=103.1667, that means UBRR = 103 approx.
     */

    twi_init();
      PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1_0 as output, 4-7 as input

    usart_init(103);
    //_delay_ms(2000); // wait for usart init

    usart_transmit_text("ESP:restart\n");
    PORTC = 0x01;
    usart_receive_text();
    PORTC = 0x02;
    //usart_receive_text(); // initial responses
    PORTC = 0x04;

    char* status = "OK\n";

    while (1) {
        char temporary = keypad_to_ascii();
        if(temporary != 0) { // when some key is pressed
            char state = temporary;
            do { // while pressed do nothing
                temporary = keypad_to_ascii();
                _delay_ms(10);
            } while(temporary == state);
            if (state == '#') status = "OK\n";
            else if (state == '9') status = "NURSECALL\n";
        }

        lcd_command(0x08); // disable lcd
        uint16_t temperature = read_temp();
        DDRD = 0xFF;

        ADCSRA |= (1<<ADSC); // Set ADSC flag of ADCSRA
                             // enable conversion
```

```
while((ADCSRA & (1<<ADSC)) == (1<<ADSC));
        // wait until flags become zero
        // that means that the conversion is complete

        lcd_init(); // init lcd
        _delay_ms(2); // wait for lcd init

        send_packet(temperature, read_adc(), status);
    }
}
```