ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2024-2025

ΑΘΗΝΑ 22 Νοεμβρίου 2024

8η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΓΙΑ ΤΟ ΜΑΘΗΜΑ “Εργαστήριο Μικροϋπολογιστών”

ΟΜΑΔΑ 23

<u>Συνεργάτες</u>

Νικόλαος Αναγνώστου 03121818

Νικόλαος Λάππας 03121098

Ζήτημα 8.1: Στην συγκεκριμένη άσκηση καλούμαστε να υλοποιήσουμε κώδικα σε C, ο οποίος να στέλνει μηνύματα στην USART και θα συνδέεται μέσω του ESP8266 στο δίκτυο του εργαστηρίου. Το πρόγραμμα περιέχει τις συναρτήσεις για την αρχικοποίηση της usart επικοινωνίας, της LCD με χρήση του PCA9555 καθώς και τρεις επιπλέον με τις οποίες κάνουμε τα εξής:

- Με την $transmit\_command(char * data)$ στέλνουμε ένα buffer που περιέχει την πληροφορία που θέλουμε να στείλουμε (παραπάνω από 1 byte συνήθως) και η συνάρτηση καλεί για κάθε χαρακτήρα του buffer την συνάρτηση $usart\_transmit(data[i])$ προκειμένου να στείλουμε την πληροφορία ανά byte.
- Με την $receive\_response(char * response)$ κάνουμε το δυαδικό ανάλογο με πριν, δεχόμαστε, δηλαδή, την πληροφορία που μας στέλνει ο server μέσω της συνάρτησης $usart\_receive()$ και την αποθηκεύουμε byte προς byte σε έναν δικό μας buffer προκειμένου να είμαστε σε θέση να το διαβάσουμε και να το τυπώσουμε. Έχουμε κάνει τις απαραίτητες ρυθμίσεις ώστε να αγνοούμε τον χαρακτήρα ‘ ” ’ και κάθε μήνυμα να λήγει με τον χαρακτήρα ‘ \n ’.
- Τέλος, με την $lcd\_print(const\ char *\ str)$ στέλνουμε όλο το μήνυμα που θέλουμε να τυπώσουμε σε μορφή string, και αυτή καλεί την $lcd\_data()$ και τυπώνει στην οθόνη έναν προς έναν τους χαρακτήρες.

Έτσι, στην $main()$ κάνουμε δυο ελέγχους μέσα σε επαναλαμβανόμενο βρόχο (μέχρι να επαληθευτεί με μήνυμα Success). Αρχικά, θέλουμε να συνδεθούμε στον server και στέλνουμε την εντολή $"ESP\!:connect\backslash n"$. Περιμένουμε για την απάντηση από την server και αν αυτή διαφέρει από το $"Success\backslash n"$ τυπώνουμε αποτυχία και επαναλαμβάνουμε. Όταν στείλει επιτυχία, στέλνουμε την εντολή $"ESP\!:url\!:\backslash"http\!://192.168.1.250\!:5000/data\backslash n\backslash""$ προκειμένου να θέσουμε το url στην κατάλληλη διεύθυνση και αναμένουμε το μήνυμα επιτυχίας ή αποτυχίας. Τα μηνύματα αυτά τα εκτυπώνουμε στην LCD οθόνη. Ακολουθεί ο κώδικας με τις νέες μόνο συναρτήσεις και την main():

```
#define F_CPU 16000000UL

#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include<stdbool.h>
#include<stdio.h>
#include<string.h>
//------------------------------- USART --------------------------------------
/* 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); // enable receiving and transmitting
   UBRR0H=(unsigned char)(ubrr>>8);
   UBRR0L=(unsigned char)ubrr;
   UCSR0C=(3 << UCSZ00); // configure the frame size to 8 data
   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;
}


void transmit_command(char *data) {
   int i = 0;
   while (data[i] != '\0') {
      usart_transmit(data[i]);
      i++;
   }
}

void receive_response(char *response) {
   char input;
   int i = 0;
```

```
    while(1){
        input = usart_receive();
        if(input == '\n') {
            response[i] = input;
            i++;
            break;
        } else {
            if (input == '"');

            else {
                response[i] = input;
                i++;
            }
        }
    }
    for(i; i<=9; i++){
        response[i] = " ";
    }
}

void lcd_print(const char* str) {
    while(*str) {
        lcd_data(*str++); // Send each character to the LCD
    }
}

#define SIZE 10
int main() {
    DDRC = 0x00;

    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); // EXT_PORT0 -> output

    usart_init(103); // for baud rate 9600
    lcd_init();

    lcd_clear_display();

    // Waiting till ESP connects ...
    while(1) {
        transmit_command("ESP:connect\n");

        char answer[SIZE];
        receive_response(answer);

        if (strcmp(answer, "Success\n") == 0) {

            lcd_print("1.Success");
            _delay_ms(2000);

            lcd_clear_display();
            break;
        }
        else lcd_print("1.Fail");
        _delay_ms(2000);

        lcd_clear_display();
    }

    // Sending command for url ...
    while(1) {
        transmit_command("ESP:url:\"http://192.168.1.250:5000/data\n\"");

        char answer_no2[SIZE];
        receive_response(answer_no2);

        if (strcmp(answer_no2, "Success\n") == 0) {
            lcd_print("2.Success");
            _delay_ms(2000);

            lcd_clear_display();
            break;
        }
        else lcd_print("2.Fail");
        _delay_ms(2000);

        lcd_clear_display();
    }
}
```

Ζήτημα 8.2: Στο συγκεκριμένο ερώτημα επεκτείναμε την παραπάνω υλοποίηση προκειμένου να παίρνουμε (προσομοίωση) τιμές πίεσης και θερμοκρασίας με χρήση του POT1 και του αισθητήρα DS18B20 αντίστοιχα. Επιπλέον, προστέθηκε η αλληλεπίδραση με το keypad προκειμένου να ανανεώνεται με την χρήση του το status σε 'NURSE CALL' και ύστερα σε 'OK'. Για τον σκοπό αυτό προσθέσαμε τις συναρτήσεις και την λογική από προηγούμενες εργαστηριακές εργασίες που είχαμε υλοποιήσει. Ακολουθεί ο κώδικας με την main().

```
#define SIZE_ANSWER 10
#define SIZE_STATUS 15
#define OFFSET 13
int main() {
    DDRB = 0xFF;
    DDRC = 0x00;
```

```
twi_init();
PCA9555_0_write(REG_CONFIGURATION_0, 0x00); // EXT_PORT0 -> output

usart_init(103); // for baud rate 9600
lcd_init();

lcd_clear_display();

// Waiting till ESP connects ...
while(1) {
    transmit_command("ESP:connect\n");

    char answer[SIZE_ANSWER];
    receive_response(answer);

    if (strcmp(answer, "Success\n") == 0) {

        lcd_print("1.Success");
        _delay_ms(2000);

        lcd_clear_display();
        break;
    }
    else lcd_print("1.Fail");
    _delay_ms(2000);

    lcd_clear_display();
} // end loop for ESP connection

// Sending command for url ...
while(1) {
    transmit_command("ESP:url:\"http://192.168.1.250:5000/data\"\n");

    char answer_no2[SIZE_ANSWER];
    receive_response(answer_no2);

    if (strcmp(answer_no2, "Success\n") == 0) {
        lcd_print("2.Success");
        _delay_ms(2000);

        lcd_clear_display();
        break;
    }
    else lcd_print("2.Fail");
    _delay_ms(2000);

    lcd_clear_display();
} // end loop for ESP url
```

```
bool pressed_no3 = false;
bool pressed_hashtag = false;

while(1) { // temperature, H2O, no3, hashtag

  // Getting the temperature
  uint16_t temperature = GetTemperature();
  int result = 0;
  int dekadika = 0;

  if (temperature == 0x8000) { // NO Device 9 bits no need for extra line
    lcd_clear_display();
    lcd_print("No Device");

  }
  else {
    if ((temperature & 0x8000) > 0) { // Negative found
      temperature = ~temperature + 1;
    } //if negative convert to its value


    if((temperature&0x01)==0x01) dekadika += 625;
    temperature = temperature >> 1;
    if((temperature&0x01)==0x01) dekadika += 1250;
    temperature = temperature >> 1;
    if((temperature&0x01)==0x01) dekadika += 2500;
    temperature = temperature >> 1;
    if((temperature&0x01)==0x01) dekadika += 5000;
    temperature = temperature >> 1 ;


    for (int i = 0; i <= 6; i++){ // Integer part -> 7 bits
      if (temperature & 0x0001 > 0) result += (int)pow(2,i);
      temperature = temperature >> 1;
    } // result saves our temperature

    result += OFFSET; // Took human temperature

  }


  // Getting the ADC-PWM
  PWM_init();
  ADC_init();

  // Connection of ADC0 with POT1
  uint16_t ADC_value = ADC_conversion();      // Read POT1
```

```
int DC_VALUE = ADC_value;
//OCR1A = DC_VALUE;
_delay_ms(100);        // Small delay for better performance

int pressure = DC_VALUE * 20 / 1023; // Took human pressure


// Waiting for keypad ...
uint8_t input = 0x00;
uint8_t savor;
//bool pressed_no3, pressed_hashtag;
char status[SIZE_STATUS];

// crucial for keypad activation with PCA
PCA9555_0_write(REG_OUTPUT_0,0x00);
PCA9555_0_write(REG_OUTPUT_1,0x00);

// Till now none of buttons 3 or # is pressed, so status is 'OK'
if (pressure > 12 | pressure < 4) { snprintf(status, sizeof(status), "CHECK PRESSURE"); }
else if (result > 37 | result < 34) { snprintf(status, sizeof(status), "CHECK TEMP"); }
else if (pressed_no3 == true && pressed_hashtag == false) { snprintf(status, sizeof(status), "NURSE CALL"); }
else snprintf(status, sizeof(status), "OK");

lcd_clear_display();
lcd_print("H20:");
if (pressure/10 != 0) { lcd_data(pressure/10 + '0'); }
lcd_data(pressure%10 + '0');

lcd_print(" Temp:");
if (result/10 != 0) { lcd_data(result/10 + '0'); }
lcd_data(result % 10 + '0');

lcd_data('.');

dekadika = dekadika/1000;
lcd_data('0' + dekadika);

lcd_command(0xC0); // New line

lcd_print(&status);
_delay_ms(50);


input = check(); // Check for the first pressed key

// Begin checking for pressed buttons
if (input == 0xFF) continue;

//first digit -> 3
```

```
        else {
           savor = input;

           while(1){
              _delay_ms(15);
              input = check(); // Check again for debouncing
              if (input == savor) continue;
              else break;
           }

           if(savor == 0xB7) {
              pressed_no3 = true;  // Found digit 3
              pressed_hashtag = false;
              snprintf(status, sizeof(status), "NURSE CALL"); // Update status because symbol # found
              lcd_clear_display();
              lcd_print(&status);

              //break; // And go check for second -> #
           }
           else if (savor == 0xBE && pressed_no3 == true) {
              pressed_hashtag = true;
              pressed_no3 = false;
           }
        }

     }

  }
```

Ζήτημα 8.3:

Αυτή η άσκηση ήταν απλά η προέκταση των 8.1 8.2 όπου επι της ουσίας έπρεπε να κάνουμε ότι πριν αλλά τώρα να στέλνουμε και το payload στον server του εργαστηρίου αλλά και να περιμένουμε την απάντηση του.
Αυτό το κάναμε με συνεχή λειτουργία όπου αρχικά γινόταν το 1^o Success μετά το 2^o και αμέσως μετά γινόντουσαν όλες οι μετρήσεις και ελεγχόταν αν έχει πατηθεί το κουμπί της νοσοκόμας και το κουμπί ακύρωσης της.
Λόγω έντονων θεμάτων του server του εργαστηρίου φροντίσαμε όταν λαμβάνουμε μία παράλογη απάντηση από τον server του εργαστηρίου , δηλαδή μία διάφορη του 200 OK τότε να ξαναγίνονται τα 1 και 2 ώστε να εξασφαλίζουμε πως πράγματι έχει εγκατασταθεί σύνδεση με το server πριν μπούμε στην διαδικασία να πάρουμε μετρήσεις και ελέγχους…επίσης μετά το 2 πέρα από μετρήσεις γίνεται η κατάλληλη κατασκευή του payload και τους status...το μόνο που χρειάζεται να διευκρινιστεί εδώ είναι ότι αν ο ασθενής έχει παράλογη πίεση και παράλογη θερμοκρασία και δεν έχει πατηθεί το nurse button,τότε το status γίνεται Check Pressure και δίνεται εκεί προτεραιότητα αφού η παράλογη πίεση είναι πολύ χειρότερη από την παράλογη θερμοκρασία στην πραγματική ζωή. Τέλος να τονισθεί πως το κουμπί nurse call(3) και το κουμπί του cancel nurse call(#) πρέπει να πατηθούν ικανό χρόνο διότι λόγω όλων των μετρήσεων που εκτελούμε και της χρονικής επιβάρυνσης τους, δεν είναι δυνατό να μπορέσουμε να ανταποκριθούμε ακαριαία σε ένα πάτημα του χρήστη. Συγκεκριμένα πρέπει να μένει πατημένο μέχρι να παγώσει το lcd_panel και μόνο τότε να αφεθεί για να το διαβάσει το πρόγραμμα μας. Παρακάτω δίνεται και ο τελικό κώδικας μας συνεχής λειτουργίας:

```
#define F_CPU 16000000UL

#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include<stdbool.h>
#include<stdio.h>
#include<string.h>

// ----------------------------- For main -------------------------------------
#define SIZE_ANSWER 16
#define SIZE_STATUS 16
#define SIZE_PAYLOAD 300
#define OFFSET 13

//-------------------------- PCA ----------------------------------------------
#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)));   // Wait till TW1 sends ACK back, means job done
   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;
   // send START condition
   TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
   // wait until transmission completed
   while(!(TWCR0 & (1<<TWINT)));
   // check value of TWI Status Register.
   twi_status = TW_STATUS & 0xF8;
   if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
   // send device address
   TWDR0 = address;
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // wail until transmission completed and ACK/NACK has been received
   while(!(TWCR0 & (1<<TWINT)));
   // check value of TWI Status Register.
   twi_status = TW_STATUS & 0xF8;
   if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
   {
      return 1; // failed to access device
   }
   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 )
   {
      // send START condition
      TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

      // wait until transmission completed
      while(!(TWCR0 & (1<<TWINT)));

      // check value of TWI Status Register.
      twi_status = TW_STATUS & 0xF8;
      if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;

      // send device address
      TWDR0 = address;
      TWCR0 = (1<<TWINT) | (1<<TWEN);

      // wail until transmission completed
      while(!(TWCR0 & (1<<TWINT)));

      // check value of TWI Status Register.
      twi_status = TW_STATUS & 0xF8;
      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);

          // wait until stop condition is executed and bus released
          while(TWCR0 & (1<<TWSTO));

          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)
{
   // send data to the previously addressed device
   TWDR0 = data;
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // wait until transmission completed

   while(!(TWCR0 & (1<<TWINT)));
   if((TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1; // write failed
   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)
{
  // send stop condition
  TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
  // wait until stop condition is executed and bus released
  while(TWCR0 & (1<<TWSTO));
}

unsigned char twi_readNak(void)
{
  TWCR0 = (1<<TWINT) | (1<<TWEN);
  while(!(TWCR0 & (1<<TWINT)));

  return TWDR0;
}

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 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;
}

// ------------------------------ LCD ----------------------------------------
void write_2_nibbles(uint8_t lcd_data) {
  uint8_t temp;
```

```
   // Send the high nibble
   temp = (PCA9555_0_read(REG_OUTPUT_0) & 0x0F) | (lcd_data & 0xF0);  // Keep lower 4 bits of PIND and set high
nibble of lcd_data
   PCA9555_0_write(REG_OUTPUT_0 , temp);                         // Output the high nibble to PORTD
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) | (1 << PD3));                    // Enable pulse
high
   _delay_us(1);                          // Small delay to let the signal settle
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & ~(1 << PD3));                  // Enable pulse
low

   // Send the low nibble
   lcd_data <<= 4;                        // Move low nibble to high nibble position
   temp = (PCA9555_0_read(REG_OUTPUT_0) & 0x0F) | (lcd_data & 0xF0);  // Keep lower 4 bits of PIND and set high
nibble of new lcd_data
   PCA9555_0_write(REG_OUTPUT_0 , temp);                         // Output the low nibble to PORTD
   PCA9555_0_write(REG_OUTPUT_0 , PCA9555_0_read(REG_OUTPUT_0) | (1 << PD3));                 // Enable pulse
high
   _delay_us(1);                          // Small delay to let the signal settle
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & ~(1 << PD3));                  // Enable pulse
low
}

void lcd_data(uint8_t data)
{
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) | 0x04);            // LCD_RS = 1, (PD2 = 1) ->
For Data
   write_2_nibbles(data);      // Send data
   _delay_ms(5);
   return;
}
void lcd_command(uint8_t data)
{
    PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & 0xFB);            // LCD_RS = 0, (PD2 = 0) -
> For Instruction
   write_2_nibbles(data);      // Send data
   _delay_ms(5);
   return;
}

void lcd_clear_display()
{
   uint8_t clear_disp = 0x01;  // Clear display command
   lcd_command(clear_disp);
   _delay_ms(5);             // Wait 5 msec
   return;
}

void lcd_init() {
   _delay_ms(200);

   // Send 0x30 command to set 8-bit mode (three times)
   PCA9555_0_write(REG_OUTPUT_0,0x30);            // Set command to switch to 8-bit mode
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) | (1 << PD3));      // Enable pulse
   _delay_us(1);
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & ~(1 << PD3));     // Clear enable
   _delay_us(30);          // Wait 250 Âµs

   PCA9555_0_write(REG_OUTPUT_0,0x30);           // Repeat command to ensure mode set
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) | (1 << PD3));
   _delay_us(1);
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & ~(1 << PD3));
   _delay_us(30);

   PCA9555_0_write(REG_OUTPUT_0,0x30);            // Repeat once more
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) | (1 << PD3));
   _delay_us(1);
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & ~(1 << PD3));
   _delay_us(30);

   // Send 0x20 command to switch to 4-bit mode
   PCA9555_0_write(REG_OUTPUT_0,0x20);
   PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) | (1 << PD3));
   _delay_us(1);
    PCA9555_0_write(REG_OUTPUT_0,PCA9555_0_read(REG_OUTPUT_0) & ~(1 << PD3));
   _delay_us(30);

   // Set 4-bit mode, 2 lines, 5x8 dots
   lcd_command(0x28);

   // Display ON, Cursor OFF
   lcd_command(0x0C);

   // Clear display
   lcd_clear_display();

   // Entry mode: Increment cursor, no display shift
   lcd_command(0x06);
}

//-------------------------------- USART ---------------------------------------
/* 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); // enable receiving and transmitting
   UBRR0H=(unsigned char)(ubrr>>8);
   UBRR0L=(unsigned char)ubrr;
   UCSR0C=(3 << UCSZ00); // configure the frame size to 8 data
   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;
}


void transmit_command(char *data) {
   int i = 0;
   while (data[i] != '\0') {
      usart_transmit(data[i]);
      i++;
   }
}

void receive_response(char *response) {
   char input;
```

```
    int i = 0;

    while(1){
        input = usart_receive();
        if(input == '\n') {
            response[i] = '\n';
            i++;
            break;
        } else {
            if (input == '"');

            else {
                response[i] = input;
                i++;
            }
        }
    }
    // Put with force spaces instead of null
    for(i; i < 16; i++){
        response[i] = '\0';
    }
}

void lcd_print(char* str) {
    while(*str) {
        lcd_data(*str++); // Send each character to the LCD
    }
}

void remove_newline(char *str) {
    size_t len = strlen(str); // Get the length of the string
    if (len > 0 && str[len - 1] == '\n') { // Check if the last character is '\n'
        str[len - 1] = '\0'; // Replace '\n' with '\0'
    }
}

//------------------------------- DS18B20 --------------------------------------
// Returns true if a connected device is found (PD4 = 0)
bool one_wire_reset()
{
    DDRD |= (1 << PD4);           // Set PD4 as output
    PORTD &= ~(1 << PD4);          // Clear PD4
    _delay_us(480);        // Delay 480 usec

    DDRD &= ~(1 << PD4);           // Set PD4 as input
    PORTD &= ~(1 << PD4);          // Disable pull-up resistor
    _delay_us(100);        // Delay 100 usec

    uint8_t input = PIND & (1 << PD4);   // Read input
    _delay_us(380);        // Delay 380 usec

    // If device is detected (PD4 = 0) -> return true
    if (input == 0x10) {return false;} // PD4 = 1
    return true;                       // PD4 = 0
}

uint8_t one_wire_receive_bit()
{
    DDRD |= (1 << PD4);           // Set PD4 as output
    PORTD &= ~(1 << PD4);         // Clear PD4
    _delay_us(2);                 // Delay 2 usec

    DDRD &= ~(1 << PD4);          // Set PD4 as input
    PORTD &= ~(1 << PD4);         // Disable pull-up resistor
    _delay_us(10);                // Delay 10 usec

    uint8_t bit_to_receive = (PIND & (1 << PD4)) ? 1 : 0;
    _delay_us(49);                // Delay 49 usec

    return bit_to_receive;
}

void one_wire_transmit_bit(uint8_t bit_to_transmit)
{
    DDRD |= (1 << PD4);           // Set PD4 as output
    PORTD &= ~(1 << PD4);         // Clear PD4
    _delay_us(2);                 // Delay 2 usec

    //PORTD |= (bit_to_transmit & 0x10);   // Send PD4 bit to connected device
    PORTD = (PORTD & ~(1 << PD4)) | ((bit_to_transmit & 0x01) ? (1 << PD4) : 0);
    _delay_us(58);                // Delay 58 usec

    DDRD &= ~(1 << PD4);          // Set PD4 as input
    PORTD &= ~(1 << PD4);         // Disable pull-up resistor
    _delay_us(1);                 // Delay 1 usec
}

uint8_t one_wire_receive_byte()
{
    uint8_t received_byte = 0x00;      // Store the byte (8-bit) we received
    for (uint8_t i = 0; i < 8; i++)
    {
        uint8_t received_bit = one_wire_receive_bit();
        received_byte |= (received_bit << i);          // Logical shift left, because DS18B20 send LSB first
        // Logical OR to insert new bit into byte sequence
    }
    return received_byte;
}
```

```
void one_wire_transmit_byte(uint8_t byte_to_transmit)
{
   for (uint8_t i = 0; i < 8; i++)
   {
      uint8_t send_bit = (byte_to_transmit >> i) & 0x01;// Bit to transmit now in position bit 0
      one_wire_transmit_bit(send_bit);
   }
}

int16_t GetTemperature()
{
   bool connected_device = one_wire_reset();   // Check for connected device
   if (!connected_device) return 0x8000;       // Error in connection return 0x8000

   one_wire_transmit_byte(0xCC);              // Only one device
   one_wire_transmit_byte(0x44);              // Begin counting temperature

   while (!one_wire_receive_bit());     // Wait until the above counting terminates

   one_wire_reset();                         // Re-initialize
   one_wire_transmit_byte(0xCC);
   one_wire_transmit_byte(0xBE);              // Read 16-bit result of temperature value

   uint16_t temperature = 0;
   temperature |= one_wire_receive_byte();     // 8-bit LSB of the total 16-bit value
   // Shift the 8-bit value 8 times to the left, OR with the previous 8-bit value
   // And take the temperature value of 16-bit
   temperature |= ((uint16_t)one_wire_receive_byte() << 8);

   return temperature;
}

//-------------------------------- POT1 ---------------------------------------
void PWM_init()
{
   // Initialize TMR1A in fast PWM 8-bit mode with non-inverted output
   // Prescaler = 1, to get 62.5kHz waveform in PB1
   TCCR1A = (1 << WGM10) | (1 << COM1A1);
   TCCR1B = (1 << CS10) | (1 << WGM12);
   // I have deleted the (0 << ... ) i had in assembly code, non necessary ones
}

void ADC_init()
{
   /* Chose ADC channel (ADC0) to read from POT1, ends in ...0000
    * For voltage reference selection: REFS0 = 1, REFS1 = 0
    * Right adjustment: ADLAR = 0
    * ADC0: MUX3 = 0, MUX2 = 0, MUX1 = 0, MUX0 = 0
```

```
     */
    ADMUX = (1 << REFS0);
    // Same as the above ADMUX = 0b01000000;

    /* Enable ADC: ADEN = 1
     * No conversion from analog to digital yet: ADSC = 0
     * Disable ADC interrupt: ADIE = 0
     * Prescaler: f_ADC = 16MHz / prescaler and
     * 50kHz <= f_ADC <= 200kHz for 10-bits accuracy. So,
     * division factor = 128 -> gives f_ADC = 125kHz
    */
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); ;
    // Same as the above ADCSRA = 0b10000111;
}

// Read the DC voltage from PB1_PWM analog filter, and take the digital result
uint16_t ADC_conversion()
{
    ADCSRA |= (1 << ADSC);          // Start conversion from analog to digital
    while(ADCSRA & (1 << ADSC));    // Wait for conversion to end
    return ADC;                     // Return the value, (ADCH:ADCL)
}

//-------------------------------- Keypad --------------------------------------
uint8_t check(){

    uint8_t stili,grammi;
    //check which stili is pressed

    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
    stili = PCA9555_0_read(REG_INPUT_1);
    if(stili == 0xF0) return 0xFF;


    PCA9555_0_write(REG_CONFIGURATION_1, 0x0F);
    grammi = PCA9555_0_read(REG_INPUT_1);
    if (grammi == 0x0F) return 0xFF;

    return ((grammi|0xF0) & (stili|0x0F));
}

int main() {
    DDRB = 0xFF;
    DDRC = 0x00;

    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); // EXT_PORT0 -> output

    usart_init(103); // for baud rate 9600
```

```
lcd_init();

lcd_clear_display();

bool pressed_no3 = false;
bool pressed_hashtag = false;

bool is_connected = false;

while(1) {
  // Waiting till ESP connects ...
  if(is_connected == false){
  while(1) {
    transmit_command("ESP:connect\n");

    char answer[SIZE_ANSWER];
    receive_response(answer);

    if (strcmp(answer, "Success\n") == 0) {
      lcd_clear_display();
      lcd_print("1.Success");
      _delay_ms(1000);

      lcd_clear_display();
      break;
    }
    else lcd_print("1.Fail");
    _delay_ms(1000);

    lcd_clear_display();
  } // end loop for ESP connection

  // Sending command for url ...
  while(1) {
    transmit_command("ESP:url:\"http://192.168.1.250:5000/data\"\n");

    char answer_no2[SIZE_ANSWER];
    receive_response(answer_no2);

    if (strcmp(answer_no2, "Success\n") == 0) {
      lcd_print("2.Success");
      _delay_ms(1000);

      lcd_clear_display();
      break;
    }
    else lcd_print("2.Fail");
    _delay_ms(1000);
    lcd_clear_display();
} // end loop for ESP url

is_connected = true;
}

// Getting the temperature
uint16_t temperature = GetTemperature();
int result = 0;
int dekadika = 0;

if (temperature == 0x8000) { // NO Device 9 bits no need for extra line
    lcd_clear_display();
    lcd_print("No Device");

}
else {
    if ((temperature & 0x8000) > 0) { // Negative found
        temperature = ~temperature + 1;
    } //if negative convert to its value


    if((temperature&0x01)==0x01) dekadika += 625;
    temperature = temperature >> 1;
    if((temperature&0x01)==0x01) dekadika += 1250;
    temperature = temperature >> 1;
    if((temperature&0x01)==0x01) dekadika += 2500;
    temperature = temperature >> 1;
    if((temperature&0x01)==0x01) dekadika += 5000;
    temperature = temperature >> 1 ;


    for (int i = 0; i <= 6; i++){ // Integer part -> 7 bits
        if (temperature & 0x0001 > 0) result += (int)pow(2,i);
        temperature = temperature >> 1;
    } // result saves our temperature

    result += OFFSET; // Took human temperature

}


// Getting the ADC-PWM
PWM_init();
ADC_init();

// Connection of ADC0 with POT1
uint16_t ADC_value = ADC_conversion();      // Read POT1
int DC_VALUE = ADC_value;
```

```
_delay_ms(10);        // Small delay for better performance

int pressure = (DC_VALUE * 20) / 1023; // Took human pressure
int pressure1 = ((DC_VALUE * 20) % 1023)/102;


// Waiting for keypad ...
uint8_t input = 0x00;
uint8_t savor;

char status[SIZE_STATUS];

// crucial for keypad activation with PCA
PCA9555_0_write(REG_OUTPUT_0,0x00);
PCA9555_0_write(REG_OUTPUT_1,0x00);

// Till now none of buttons 3 or # is pressed, so status is 'OK'
if (pressed_no3 == true && pressed_hashtag == false) { snprintf(status, sizeof(status), "NURSE CALL"); }
else if (pressure > 12 | pressure < 4) { snprintf(status, sizeof(status), "CHECK PRESSURE"); }
else if (result > 37 | result < 34) { snprintf(status, sizeof(status), "CHECK TEMP"); }
else snprintf(status, sizeof(status), "OK");

lcd_clear_display();
lcd_print("H20:");
if (pressure/10 != 0) { lcd_data(pressure/10 + '0'); }
lcd_data(pressure%10 + '0');

lcd_print(" Temp:");
if (result/10 != 0) { lcd_data(result/10 + '0'); }
lcd_data(result % 10 + '0');

lcd_data('.');

lcd_data('0' + dekadika/1000);

lcd_command(0xC0); // New line

lcd_print(&status);
_delay_ms(40);


input = check(); // Check for the first pressed key

// Begin checking for pressed buttons
if (input == 0xFF);

//first digit -> 3
else {
  savor = input;
```

```
        while(1){
            _delay_ms(15);
            input = check(); // Check again for debouncing
            if (input == savor) continue;
            else break;
        }

        if(savor == 0xB7) {
            pressed_no3 = true;  // Found digit 3
            pressed_hashtag = false;
            snprintf(status, sizeof(status), "NURSE CALL"); // Update status because symbol # found
            lcd_clear_display();
            lcd_print(&status);
        }

        else if (savor == 0xBE && pressed_no3 == true) {
            pressed_hashtag = true;
            pressed_no3 = false;
        }
    }
    _delay_ms(10); // Delay to take the keys

    dekadika = dekadika / 1000;

    char message[SIZE_PAYLOAD];
    snprintf(message,                                                          sizeof(message),
"ESP:payload:[{\"name\":\"temperature\",\"value\":\"%d.%d\"},{\"name\":\"pressure\",\"value\":\"%d.%d\"},{\"n
ame\":\"team\",\"value\":\"23\"},{\"name\":\"status\",\"value\":\"%s\"}]\n",         result,          dekadika,
pressure,pressure1,status);
    transmit_command(message);
    //_delay_ms(2000);

    char answer_no3[SIZE_ANSWER];
    receive_response(answer_no3);

    if (strcmp(answer_no3, "Success\n") == 0) {
        lcd_clear_display();
        lcd_print("3.Success");
        _delay_ms(40);

        transmit_command("ESP:transmit\n");

        char answer_no4[SIZE_ANSWER];
        receive_response(answer_no4);

        if (strcmp(answer_no4, "200 OK\n") == 0) {
            lcd_clear_display();
            //lcd_print("4. 200 OK");
            remove_newline(answer_no4);

            lcd_print("4. ");
            lcd_print(answer_no4);
            _delay_ms(40);

         }
         else {
            lcd_clear_display();
            //lcd_print("4.Fail");

            remove_newline(answer_no4);

            lcd_print("4. ");
            lcd_print(answer_no4);
            is_connected = false;
         }
         _delay_ms(40);

      }
      else
      {
         lcd_clear_display();
         lcd_print("3.Fail");
         _delay_ms(40);
         is_connected = false;
      }
   }
}
```