2024

Αναφορά 8ης Εργαστηριακής Άσκησης Εργαστηρίου Μικροϋπολογιστών

ΞΈΝΟΣ ΕΜΜΑΝΟΥΗΛ 03120850
ΠΑΠΑΔΟΠΟΥΛΟΣ ΣΠΥΡΙΔΩΝ 03120033

Προτού προχωρήσουμε στην ανάλυση της του προγράμματος για την επικοινωνία του ntuAboard με τον server θα πρέπει να αρχικά να δώσουμε μία σύντομη περιγραφή των συναρτήσεων που χρησιμοποιήσαμε ώστε να επιτευχθεί αυτή η επικοινωνία.

```
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;
}
```

Αρχικά έχουμε την συνάρτηση usart_init η οποία παίρνει ως argument το UBRR και ουσιαστικά μας επιτρέπει να έχουμε ασύγχρονη επικοινωνία με τον server, πραγματοποιώντας αποστολή και λήψη μηνυμάτων των 8bit. Το argument UBRR μας βοηθάει να ρυθμίσουμε το επιθυμητό BAUD χρησιμοποιώντας την εξίσωση:

$$U\,BRR = \frac{f_{osc}}{16\,BAUD} - 1$$

Όπου f_{osc} είναι η συχνότητα λειτουργίας στον μικροελεγκτή που χρησιμοποιούμε.

Στην συνέχεια έχουμε την συνάρτηση που μας επιτρέπει να στέλνουμε δεδομένα στον server.

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

void usart_transmit_string(const char str[]){
    int i;
    for(i=0; str[i]!='\0'; i++) usart_transmit(str[i]);
}
```

Η συνάρτηση usart_transmit_string παίρνει ως όρισμα ένα string και μέσα σε αυτή στέλνουμε έναν-έναν τους χαρακτήρες του μέχρι το end of string που συμβολίζεται στην C με \0. Η συνάρτηση usart_transmit αποθηκεύει τα bytes δεδομένων στον καταχωρητή UDR0 από τον οποίο στέλνονται τα δεδομένα από το UART. Ουσιαστικά σε αυτή την συνάρτηση περιμένουμε μέχρι ο καταχωρητής αποστολής δεδομένων UDR0 να είναι έτοιμος να δεχθεί νέα δεδομένα, που σηματοδοτείται με το όταν η σημαία UDRE0 είναι 1.

Η συνάρτηση που χρησιμοποιούμε για να λάβουμε ένα string από δεδομένα από τον server είναι:

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

```
void usart_receive_string(char string[]) {
    int i = 0;
    char received;
    while((received = usart_receive()) != '\n') {
        string[i++] = received;
    }
    string[i] = '\0';
}
```

Στην συνάρτηση usart_receive_string γράφουμε την πληροφορία που λάβαμε σε ένα string που το παίρνουμε ως όρισμα. Σε αυτό γράφουμε έως ότου ο χαρακτήρας που λαμβάνουμε είναι το \n, όπου στην επικοινωνία μας σε αυτή την άσκηση σημαίνει ότι τελείωσε το μήνυμα. Για να μπορέσουμε αυτό το string να το χρησιμοποιήσουμε σωστά στην C προσθέτουμε μετά το \n και το \0 που σημαίνει ότι εδώ τελειώνει το string. Η συνάρτηση usart_receive που χρησιμοποιείται στην usart_receive_string περιμένει μέχρι να υπάρχουν δεδομένα προς ανάγνωση και στην συνέχεια επιστρέφει το περιεχόμενο του καταχωρητή UDR0. Το ότι υπάρχουν δεδομένα προς ανάγνωση σηματοδοτείται από το όταν η σημαία RXC0 είναι 1.

Για την κατασκευή του payload που θα σταλθεί στον server χρησιμοποιείται η ακόλουθη συνάρτηση που το κατασκευάζει σταδιακά.

```
char payload[168];

void append_to_payload(const char *key, const char *value) {
    strcat(payload, "{\"name\": \"");
    strcat(payload, key);
    strcat(payload, "\",\"value\": \"");
    strcat(payload, value);
    strcat(payload, "\"},");
}
```

Έχουμε λοιπόν αναλύσει όλες τις συναρτήσεις που χρειαζόμαστε για την επικοινωνία με τον server καθώς και όποιες άλλες χρειαζόμαστε, επομένως μπορούμε να προχωρήσουμε στην υλοποίηση της εργασίας. Συναρτήσεις που δεν έχουν αναφερθεί παραπάνω έχουν υλοποιηθεί σε προηγούμενες εργαστηριακές ασκήσεις και θα παρατεθούν στο τέλος της αναφοράς ώστε το κείμενο να είναι πιο ευανάγνωστο, αλλά ταυτόχρονα να υπάρχει η υλοποίησή τους σε περίπτωση που χρειαστεί.

Η συνάρτηση main είναι η ακόλουθη:

```
int main(){

    twi_init();

    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);

    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1[7:4] as
input and EXT_PORT[3:0] as output

    ADMUX |= 0b01000000; //Right adjusted, ADC0
```

```
    ADCSRA |= 0b10000111;   //128 Prescaler

    char recMes[20];

    lcd_init();

    usart_init(103);
    usart_transmit_string("ESP:restart\n");
    usart_receive_string(recMes);

    int nurse_call=0;

    uint8_t failed, first;
    first=0;
    while(1){

        lcd_clear_display();
        do {
            char receivedMessage[30];
            failed=0;
            lcd_clear_display();
            usart_transmit_string("ESP:connect\n");
            usart_receive_string(receivedMessage);
            if(strcmp(receivedMessage, "\"Success\"")==0){
              lcd_string("1.Success");
            }else {
                lcd_string("1.Fail");
                failed=1;
            }
           _delay_ms(1000);
        } while(failed);

        do {
            char receivedMessage2[30];
            failed=0;
            lcd_clear_display();
            if(!first) {
                usart_receive_string(receivedMessage2);
                first=1;
            }

usart_transmit_string("ESP:url:\"http://192.168.1.250:5000/data\"\n");
            usart_receive_string(receivedMessage2);
            if(strcmp(receivedMessage2, "\"Success\"")==0){
              lcd_string("2.Success");
            }else {
                lcd_string("2.Fail");
                failed=1;
```

```
            }
           _delay_ms(1000);
        } while(failed);

        lcd_clear_display();


        //Temperature calculation
        char temperature[6];
        calculateTemp(temperature);


        //Calculate Pressure
        ADCSRA|= (1<<ADSC);    //Start ADC
        while(ADCSRA & (1<<ADSC)); //Wait until ADC is finished
        uint8_t pressureNum=ADC/50;

        char pressureStr[2];
        dtostrf(pressureNum, 3, 2, pressureStr);

        //Checking status
        char *status;
        lcd_string("Checking status...");
        _delay_ms(1500);
        lcd_clear_display();

        status="OK";

        if(keypad_to_ascii()=='5' || nurse_call==1){
            status="NURSECALL";
            nurse_call=1;
            _delay_ms(1500);
        }

        if(keypad_to_ascii()=='#'){
            status="OK";
            nurse_call=0;
        }

        if((pressureNum>12 || pressureNum<4) && !nurse_call)
status="CHECKPRESSURE";
        if((temp_num>37 || temp_num<34) && !nurse_call)
status="CHECKTEMP";

        //writing status, temp and pressure in lcd

        lcd_string(temperature);
        lcd_data(0xDF);
        lcd_string("C ");
```

```
        lcd_string(pressureStr);
        lcd_string("cm H20");
        lcd_command(0xC0);
        lcd_string(status);

        _delay_ms(2500);
        lcd_clear_display();

        strcpy(payload, "ESP:payload:[");
        append_to_payload("temperature", temperature);
        append_to_payload("pressure", pressureStr);
        append_to_payload("team", "45");
        append_to_payload("status", status);
        payload[strlen(payload) - 1] = '\0';
        strcat(payload, "]\n");
        */
        do{
            failed=0;
            char receivedMessage2[30];
            lcd_clear_display();
            usart_transmit_string(payload);
            usart_receive_string(receivedMessage2);
            if(strcmp(receivedMessage2, "\"Success\"")==0){
                lcd_string("3.Success");
            }else {
                  lcd_string("3.Fail");
                  failed=1;
            }
            _delay_ms(2000);
        }while(failed);

        lcd_clear_display();
        char end[30];
        usart_transmit_string("ESP:transmit\n");
        usart_receive_string(end);
        lcd_string("4.");
        lcd_string(end);

        _delay_ms(2000);
        lcd_clear_display();

    }


}
```

Στην αρχή γίνεται αρχικοποίηση του πρωτοκόλλου TWI και θέτουμε την θύρα επέκτασης 0 του PCA9555 ως έξοδο, ενώ στην θύρα επέκτασης 1 του PCA9555 θέτουμε τα 4 MSB ως είσοδο και το 4 LSB ως έξοδο. Αυτές οι τιμές επιλέχθηκαν καθώς στην θύρα επέκτασης 0 είναι συνδεδεμένη η LCD οθόνη και στην θύρα

επέκτασης 1 το 4x4 πληκτρολόγιο. Έπειτα ρυθμίζουμε κατάλληλα το ADC0 της πλακέτας, αρχικοποιούμε την οθόνη και το UART και του κάνουμε επανεκκίνηση. Η τιμή 103 στην αρχικοποίηση του UART προκύπτει από το ότι έχουμε μικροελεγκτή συχνότητας 16MHz και θέλουμε BAUD 9600.

Μετά από τις απαραίτητες αρχικοποιήσεις μπαίνουμε σε έναν ατέρμονα βρόχο ο οποίος αποτελείται από 4 στάδια. Σε κάθε στάδιο εκτός από το τελευταίο επιμένουμε έως ότου λάβουμε “Success” από τον server.

Στο πρώτο στάδιο προσπαθούμε να συνδεθούμε στον server στέλνοντας το μήνυμα “ESP:connect\n”. Μέχρι να λάβουμε “Success” παραμένουμε σε αυτό το στάδιο. Αν λάβουμε “Success” εκτυπώνουμε στην οθόνη “1.Success” και πηγαίνουμε στο δεύτερο στάδιο όπου στέλνουμε “ ESP:url:"http://192.168.1.250:5000/data” και επαναλαμβάνουμε μέχρι να λάβουμε “Success” από τον server. Σε περίπτωση που λάβουμε “Fail” στο στάδιο 1 εκτυπώνουμε στην οθόνη “1.Fail”, ενώ αν λάβουμε “Fail” στο στάδιο 2 εκτυπώνουμε “2.Fail”. Αξίζει να σημειωθεί εδώ ότι παρατηρήθηκε κατά την διάρκεια που κάναμε την εργασία ότι ο server στην πρώτη φορά που συνδεόμασταν σε αυτόν μετά το “Success” που στέλνει στο πρώτο στάδιο φαίνεται να στέλνει άλλο ένα dummy μήνυμα και για αυτό την πρώτη μόνο φορά που μπαίνουμε στο δεύτερο στάδιο κάνουμε ένα dummy receive_string.

Αφού λοιπόν λάβαμε “Success” και στο δεύτερο στάδιο έχουμε συνδεθεί στον server και πρέπει να κατασκευάσουμε το payload που πρόκειται να στείλουμε. Για αυτό μετράμε την θερμοκρασία μέσω της calculateTemp και παίρνουμε την πίεση από το POT0 κάνοντας ADC και διαιρώντας την τιμή που λάβαμε με 50. Το 50 προκύπτει από το ότι έχουμε 10bit ADC, άρα μέγιστη τιμή 1024, και η μέγιστη τιμή της πίεσης είναι 20 cm H_20 και $\frac{1024}{20} \cong 50$, και μετατρέπουμε την τιμή της πίεσης σε string. Στην συνέχεια ελέγχουμε το status του ασθενή. Αν ο ασθενής πατήσει το πλήκτρο 5 (το τελευταίο ψηφίο του αριθμού της ομάδος μας) εκείνη την στιγμή ή έχει πατήσει το 5 σε προηγούμενη επανάληψη του βρόχου τότε το status του γίνεται “NURSECALL”. Αν πατήσει την δίεση τότε το status του από NURSECALL γίνεται OK. Αν η πίεση δεν είναι μεταξύ των τιμών 4 με 12 και δεν έχει πατηθεί το NURSECALL τότε το status του γίνεται CHECKPRESSURE. Αν η θερμοκρασία του δεν είναι ανάμεσα σε 34 με 37 βαθμούς και δεν έχει πατηθεί το NURSECALL τότε το status γίνεται CHECKTEMP. Αν δεν έχει πατηθεί NURSECALL (ή πατήθηκε προηγουμένως και τώρα ακυρώθηκε) και η θερμοκρασία και η πίεση του ασθενή είναι «καλή» τότε το status του γίνεται OK. Επειδή η εκφώνηση δεν επιτρέπει να υπάρχει status που να ειδοποιεί και για έλεγχο της θερμοκρασίας και της πίεσης, σε περίπτωση που χρειάζονται να γίνουν και τα δύο το status που μένει είναι η τελευταία τιμή του status, δηλαδή το CHECKTEMP, επειδή ο έλεγχος φυσιολογικής τιμής της θερμοκρασίας γίνεται μετά της πίεση. Αφού έχουμε όλες τις αναγκαίες πληροφορίες για το payload τις δείχνουμε την οθόνη και κατασκευάζουμε το payload. Όταν το payload κατασκευαστεί το στέλνουμε συνεχώς στον server μέχρι να λάβουμε το μήνυμα “Success” από αυτόν.

Στο τέταρτο στάδιο αφού έχουμε στείλει το payload στέλνουμε το μήνυμα “ ESP:transmit” στον server και εκτυπώνουμε την απάντηση του server στην οθόνη. Αν όλα έχουν πάει καλά με το payload λαμβάνουμε απάντηση 200 OK.

Ακολουθεί ο πλήρης κώδικας της υλοποίησής μας:

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <avr/pgmspace.h>
#define PCA9555_0_ADDRESS 0x40
#define TWI_READ 1
#define TWI_WRITE 0
#define SCL_CLOCK 100000L
//A0=A1=A2=0 by hardware
// reading from twi device
// writing to twi device
// 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;

}

//Read one byte from the twi device, read is followed by a stop
condition
unsigned char twi_readNak(void){

    TWCR0 = (1<<TWINT) | (1<<TWEN);
    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; // 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)));

    // 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;
    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;
    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));
}

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;
}
```

```
//function to get the number of row to check if a button is pressed.
Returns an 8 bit int with the buttons that were pressed.
uint8_t scan_row(uint8_t row){
    uint8_t check=1;
    check = check << row;
    PCA9555_0_write(REG_OUTPUT_1, ~check);
    uint8_t result = PCA9555_0_read(REG_INPUT_1)>>4;
    return result;
}

//function to check the entire 4x4 keypad. Returns an 16 bit int with
the buttons that were pressed.
//The formation of the result is r0_r1_r2_r3 where its r0 is 4 bits
IO1_7, IO1_6, IO_5, IO_4
uint16_t scan_keypad(){
    uint16_t result;
    result= (scan_row(0)<<12) + (scan_row(1)<<8) + (scan_row(2)<<4)
+scan_row(3);
    return result;
}

uint8_t previous_call=0;

uint16_t scan_keyboard_rising_edge(){
    uint16_t result0 = scan_keypad();
    _delay_ms(15);
    uint16_t result1 = scan_keypad();
    uint16_t same = (~result0 & ~result1);
    previous_call=~same;
    return same;
}


uint8_t keypad_to_ascii(){

        switch(scan_keyboard_rising_edge()) {
        case 0: return 0;
        case 1: return '1';
        case 2: return '2';
        case 4: return '3';
        case 8: return 'A';
        case 16: return '4';
        case 32: return '5';
        case 64: return '6';
        case 128: return 'B';
        case 256: return '7';
        case 512: return '8';
        case 1024: return '9';
        case 2048: return 'C';
```

```
        case 4096: return '*';
        case 8192: return '0';
        case 16384: return '#';
        case 32768: return 'D';
    }
        return 0;
}

void write_2_nibbles(uint8_t c);
void lcd_clear_display();
void lcd_command(uint8_t com);
void lcd_data(unsigned char data);
void lcd_init();
void lcd_string(const char *str);

void write_2_nibbles(uint8_t c){

    uint8_t temp= c;
    PCA9555_0_write(REG_OUTPUT_0, (PCA9555_0_read(REG_INPUT_0) & 0x0f)
+ (temp & 0xf0)); //LCD Data High Bytes
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
    asm("nop");
    asm("nop");
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) &
(~0x08));

    c=(c<<4)|(c>>4);
    PCA9555_0_write(REG_OUTPUT_0, (PCA9555_0_read(REG_INPUT_0) & 0x0f)
+ (c & 0xf0)); //LCD Data Low Bytes

    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
    asm("nop");
    asm("nop");
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) &
(~0x08));

}

void lcd_clear_display(){
    lcd_command(0x01);
    _delay_ms(5);
}

void lcd_command(uint8_t com){
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) &
(~0x04)); //LCD_RS=0 => Instruction
    write_2_nibbles(com);
    _delay_us(250);
```

```
}

void lcd_data(uint8_t  data){
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) |
0x04); //LCD_RS=1 => Data
    write_2_nibbles(data);
    _delay_us(250);
}

void lcd_init(){
    _delay_ms(200);
    int i=0;
    while(i<3){      //command to switch to 8 bit mode
        PCA9555_0_write(REG_OUTPUT_0, 0x030);
        PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) |
0x08);
        asm("nop");
        asm("nop");
        PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) &
(~0x08));
        _delay_us(250);
        ++i;
    }

    PCA9555_0_write(REG_OUTPUT_0, 0x20); //command to switch to 4 bit
mode
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
    _delay_us(2);
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) &
(~0x08));

    _delay_us(250);

    lcd_command(0x28); //5*8 dots, 2 lines
    lcd_command(0x0c); //display on, cursor off

    lcd_clear_display();

}

void lcd_string(const char *str){
    int i;
    for(i=0; str[i]!=0; i++) lcd_data(str[i]);
}


uint8_t one_wire_reset() {
  DDRD |= (1 << PD4);
  PORTD &= ~(1 << PD4);
```

```
  _delay_us(480);

  DDRD &= ~(1 << PD4);
  PORTD &= ~(1 << PD4);
  _delay_us(100);

  uint8_t input = PIND;
  _delay_us(380);

  return input & (1 << PD4) ? 0 : 1;
}

uint8_t one_wire_receive_bit() {
  DDRD |= (1 << PD4);
  PORTD &= ~(1 << PD4);
  _delay_us(2);

  DDRD &= ~(1 << PD4);
  PORTD &= ~(1 << PD4);
  _delay_us(10);

  uint8_t input = PIND;
  _delay_us(49);
  return input & (1 << PD4) ? 1 : 0;
}

void one_wire_transmit_bit(uint8_t data){
  DDRD |= (1 << PD4);
  PORTD &= ~(1 << PD4);
  _delay_us(2);

  PORTD|=(data<<PD4);
  _delay_us(58);

  DDRD &= ~(1 << PD4);
  PORTD &= ~(1 << PD4);
  _delay_us(1);
}

uint8_t one_wire_receive_byte(){
  uint8_t i=0;
  uint8_t receive_result=0;
  while(i<8){
    receive_result+= (one_wire_receive_bit()<<i);
    i++;
  }
  return receive_result;
}
```

```
void one_wire_transmit_byte(uint8_t data){
  uint8_t i=0;
  while(i<8){
    one_wire_transmit_bit(data & 0x01);
    data=data>>1;
    i++;
  }
}

uint8_t low, high;

void measureTemp(){
  if(!one_wire_reset()){
    high=0x080;
    low=0x00;
    __asm__ __volatile__(
    "mov r24, %1" "\n\t"
    "mov r25, %0" "\n\t"
    :
    : "r" (high), "r" (low)
    );
    return;
  }

  one_wire_transmit_byte(0xCC);

  one_wire_transmit_byte(0x44);

  while(one_wire_receive_bit()!=1);

  if(!one_wire_reset()){
    high=0x80;
    low=0x00;
    __asm__ __volatile__(
    "mov r24, %1" "\n\t"
    "mov r25, %0" "\n\t"
    :
    : "r" (high), "r" (low)
    );
    return;
  }



  one_wire_transmit_byte(0xCC);

  one_wire_transmit_byte(0xBE);
```

```
  low=one_wire_receive_byte();
  high=one_wire_receive_byte();
  __asm__ __volatile__(
    "mov r24, %1" "\n\t"
    "mov r25, %0" "\n\t"
    :
    : "r" (high), "r" (low)
    );

}
float temp_num;
void calculateTemp(char temp[]){
   measureTemp();
  lcd_clear_display();

  if(high==0x80 && low==0x00){
    lcd_string("No Device");
    return;
  }


  uint16_t temperature = (uint16_t)((high << 8) | low);

  if(high & 0xF8){ //if temperature is negative
    temperature=~temperature+1;
    lcd_data('-');
  }


  float num = temperature *0.0625*1.55;
  temp_num=num;
  dtostrf(num, 3, 2, temp);
}


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;
}

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

```
void usart_transmit_string(const char str[]){
    int i;
    for(i=0; str[i]!='\0'; i++) usart_transmit(str[i]);
}

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


void usart_receive_string(char string[]) {
    int i = 0;
    char received;
    while((received = usart_receive()) != '\n') {
        string[i++] = received;
    }
    string[i] = '\0';
}
char payload[168];

void append_to_payload(const char *key, const char *value) {
    strcat(payload, "{\"name\": \"");
    strcat(payload, key);
    strcat(payload, "\",\"value\": \"");
    strcat(payload, value);
    strcat(payload, "\"},");
}


int main(){

    twi_init();

    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);

    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1[7:4] as
input and EXT_PORT[3:0] as output


    ADMUX |= 0b01000000; //Right adjusted, ADC0
    ADCSRA |= 0b10000111;   //128 Prescaler

    char recMes[20];

    lcd_init();

    usart_init(103);
```

```
    usart_transmit_string("ESP:restart\n");
    usart_receive_string(recMes);

    int nurse_call=0;

    uint8_t failed, first;
    first=0;
    while(1){

        lcd_clear_display();
        do {
            char receivedMessage[30];
            failed=0;
            lcd_clear_display();
            usart_transmit_string("ESP:connect\n");
            usart_receive_string(receivedMessage);
            if(strcmp(receivedMessage, "\"Success\"")==0){
              lcd_string("1.Success");
            }else {
                lcd_string("1.Fail");
                failed=1;
            }
           _delay_ms(1000);
        } while(failed);

        do {
            char receivedMessage2[30];
            failed=0;
            lcd_clear_display();
            if(!first) {
                usart_receive_string(receivedMessage2);
                first=1;
            }

usart_transmit_string("ESP:url:\"http://192.168.1.250:5000/data\"\n");
            usart_receive_string(receivedMessage2);
            if(strcmp(receivedMessage2, "\"Success\"")==0){
              lcd_string("2.Success");
            }else {
                lcd_string("2.Fail");
                failed=1;
            }
           _delay_ms(1000);
        } while(failed);

        lcd_clear_display();


        //Temperature calculation
```

```
        char temperature[6];
        calculateTemp(temperature);


        //Calculate Pressure
        ADCSRA|= (1<<ADSC);    //Start ADC
        while(ADCSRA & (1<<ADSC)); //Wait until ADC is finished
        uint8_t pressureNum=ADC/50;

        char pressureStr[2];
        dtostrf(pressureNum, 3, 2, pressureStr);

        //Checking status
        char *status;
        lcd_string("Checking status...");
        _delay_ms(1500);
        lcd_clear_display();

        status="OK";

        if(keypad_to_ascii()=='5' || nurse_call==1){
            status="NURSECALL";
            nurse_call=1;
            _delay_ms(1500);
        }

        if(keypad_to_ascii()=='#'){
            status="OK";
            nurse_call=0;
        }

        if((pressureNum>12 || pressureNum<4) && !nurse_call)
status="CHECKPRESSURE";
        if((temp_num>37 || temp_num<34) && !nurse_call)
status="CHECKTEMP";

        //writing status, temp and pressure in lcd

        lcd_string(temperature);
        lcd_data(0xDF);
        lcd_string("C ");
        lcd_string(pressureStr);
        lcd_string("cm H20");
        lcd_command(0xC0);
        lcd_string(status);

        _delay_ms(2500);
        lcd_clear_display();
```

```
        strcpy(payload, "ESP:payload:[");
        append_to_payload("temperature", temperature);
        append_to_payload("pressure", pressureStr);
        append_to_payload("team", "45");
        append_to_payload("status", status);
        payload[strlen(payload) - 1] = '\0';
        strcat(payload, "]\n");
        */
        do{
            failed=0;
            char receivedMessage2[30];
            lcd_clear_display();
            usart_transmit_string(payload);
            usart_receive_string(receivedMessage2);
            if(strcmp(receivedMessage2, "\"Success\"")==0){
                lcd_string("3.Success");
            }else {
                  lcd_string("3.Fail");
                  failed=1;
            }
            _delay_ms(2000);
        }while(failed);

        lcd_clear_display();
        char end[30];
        usart_transmit_string("ESP:transmit\n");
        usart_receive_string(end);
        lcd_string("4.");
        lcd_string(end);

        _delay_ms(2000);
        lcd_clear_display();

    }


}
```