Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών
Εργαστήριο Μικροϋπολογιστών

Έκθεση 8ης Εργαστηριακής Άσκησης

Στοιχεία:
Ομάδα: **39**
Μέλη: **Ευάγγελος Μυργιώτης, Αριστοτέλης Γρίβας**

Ζητήματα 8.1,2,3 :

Η λειτουργία του κώδικα βασίζεται στις παρακάτω συναρτήσεις :

-usart_transmit_command(const char a[],int b):Η συνάρτηση αυτή καλεί την usart_transmit() (που στέλνει ένα byte στη usart) επαναλβανόμενα,στέλνοντας όλους τους χαρακτήρες του πίνακα a[].Το όρισμα b αποτελεί τον αριθμό των χαρακτήρων που έχει κάθε φορά ο πίνακας a και συνεπώς τον αριθμό των επαναλήψεων της usart_transmit().Αφού έχει στείλει όλα τα bytes,στο τέλος στέλνει και το χαρακτήρα '\n'.

-lcd_received():Λειτουργεί όπως η συνάρτηση που μόλις περιγράψαμε,αλλά αυτή τη φορά λαμβάνει ένα-ένα τα bytes που βρίσκονται στη usart(μέχρι να βρει το χαρακτήρα '\n') μέσω της usart_receive,τα αποθηκεύει σε έναν πίνακα (global πίνακας rec[]),και στη συνέχεια τα τυπώνει στην οθόνη lcd.

-usart_flash():η συνάρτηση αυτή "καθαρίζει" τον buffer των 2 bytes του ESP,ώστε στη νέα επανάληψη της lcd_received να τυπωθεί σωστά το νέο μήνυμα που λαμβάνεται από τη usart.

-buffer_clear():η συνάρτηση αυτή καλεί τη usart_flash() ώστε να καθαρίσει τον buffer του ESP και στη συνέχεια καθαρίζει και την global μεταβλητή rec[] όπου αποθηκεύουμε τα δεδομένα που λαμβάνονται.

-usart_transmit_word(const char a[],int b): έχει ακριβώς την ίδια λειτουργία με την usart_transmit_command(),αλλά στέλνει μόνο τη συμβολοσειρά,χωρίς να στέλνει το χαρακτήρα '\n' στο τέλος.

-usart_transmit_payload(const char status[],int k):η συνάρτηση αυτή στέλνει στη usart το απαιτούμενο payload.Χρησιμοποιεί τη συνάρτηση usart_transmit_word() για την επαναλαμβανόμενη αποστολή των διάφορων μηνυμάτων του payload.Επίσης,κάνει χρήση των μεταβλητών temp[],pres[],status[] όπου έχουμε αποθηκευμένα τα μεταβλητά σημεία του payload(θερμοκρασία,πίεση,status).

Τέλος όσον αφορά στον υπόλοιπο κώδικα,το διάβασμα του πληκτρολογίου και ο υπολογισμός της πίεσης και τις θερμοκρασίας,γίνονται με τον ίδιο τρόπο που περιγράψαμε στις προηγούμενες εργασίες.Η μόνη εξαίρεση είναι ότι η πίεση πολλαπλασιάζεται με το 4(ώστε να έχουμε κλίμακα 0-20 αντί για 0-5 που υπολογίζει το ποτενσιόμετρο) και η θερμοκρασία αθροίζεται με το 15 (ώστε να προσομοιώνεται η υποθετική θερμοκρασία δωματίου ίση με 36 βαθμους κελσίου).
Ο κώδικας αρχικά καλεί την usart_init(103) για να αρχικοποίησει την usart στα BAUD = 9600 που θέλουμε, και στη συνέχεια αρχίζουμε να στέλνουμε
και να λαμβάνουμε τα μυνήματά μας με την σειρά που αναφέρεται στην εκφώνηση μέσω των εντολων usart_transmit_command() και lcd_received().
Μετά από κάθε lcd_received() καλούμε τη συνάρτηση clear_buffer() για να καθαρίσουμε τους πίνακές μας.

Κώδικας :

```
.#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
//#include<string.h>
#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
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
//#define  temp r24

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

//unsigned char A,B,C,D,F1,F0;
unsigned char t1,t2;
//unsigned char x1,x2;
uint16_t x1,x2;
uint8_t z,y;
unsigned char  a0 = 0;
unsigned char  a1 = 0;
unsigned char  a2 = 0;
unsigned char  a3 = 0;
unsigned char  a4 = 0;
int v;

uint8_t  p;




int s;
uint8_t con[] = "ESP:connect";
uint8_t  rec[50];
uint8_t pres[2];
uint8_t temp[2];
int te1,te2,te3,p0,p1,p2,p3;
uint8_t status1[] = "NURSECALL";
uint8_t status2[] = "CHECKTEMP";
uint8_t status3[] = "CHECKPRESSURE";
uint8_t status4[] = "OK";

 //uint8_t status[];









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;
}

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)));
// 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;
}
void write_2_nibbles(unsigned char q){
    int temp = q;   // r24 = temp = q
    int e = PIND;   //in r25, PIND
    e = e & 0x0f;    // andi r25, 0x0f
    temp = temp & 0xf0; // andi r24, 0xf0
    temp = temp + e;    // add r24, r25
    PORTD= temp;  //out PORTD ,r24
    PORTD |= 0x08;  //sbi PORTD ,3
    _delay_ms(1);
    PORTD &= 0b11110111; //cbi PORTD ,3
    _delay_ms(1);

    temp = q;   //pop r24
    temp = temp<<4;  //swap r24
    temp = temp & 0xf0;  //andi r24, 0xf0
    temp = temp + e;   //add r24,r25
    PORTD= temp;   //portd, r24
    PORTD |= 0x08;
    _delay_ms(1);
    PORTD &= 0b11110111;
    _delay_ms(2);

return;
}


void lcd_data(unsigned char p){
    PORTD |= 0x04;
    write_2_nibbles(p);
    _delay_ms(10);
    return;
}
```

```
void lcd_command(unsigned char z){
    PORTD &= 0b11111011;
    write_2_nibbles(z);
    _delay_ms(10);
    return;
}


void lcd_init(){

    _delay_ms(40); //wait_msec

        PORTD = 0x30;     //PORTD = r24 = 0x30
        PORTD |= 0x08;   // sbi PORTD, 3
        _delay_ms(1);
        PORTD &= 0b11110111; //cbi PORTD, 3
        _delay_ms(2);

         PORTD = 0x30;     //PORTD = r24 = 0x30
        PORTD |= 0x08;   // sbi PORTD, 3
        _delay_ms(1);
        PORTD &= 0b11110111; //cbi PORTD, 3
        _delay_ms(2);

        PORTD = 0x30;   //PORTD = r24 = 0x30
        PORTD |= 0x08;
        _delay_ms(1);
        PORTD &= 0b11110111;
        _delay_ms(2);

        PORTD = 0x20;
        PORTD |= 0x08;
        _delay_ms(1);
        PORTD &= 0b11110111;
        _delay_ms(2);
        lcd_command(0x28);
        lcd_command(0x0c);
        lcd_command(0x01);

        _delay_ms(10);

        lcd_command(0x06);

        _delay_ms(10);

        return;
}


unsigned char one_wire_reset(){

            unsigned char  temp;
            DDRD |= 0b00010000;
            PORTD &= 0b11101111;
```

```
        _delay_us(480);
        DDRD &= 0b11101111;
        PORTD &= 0b11101111;
        _delay_us(100);
        temp = PIND;            //temp = r24 = PORTD
        _delay_us(380);
        if(temp & 0b00010000) {         //if PD4 = 0 return 1 in temp = r24
                temp = 0;
                return temp;
        }
        else{                   //else return 0 in temp = r24
         temp = 1;
         return temp;
        }

}


unsigned char one_wire_receive_bit(){

        unsigned char temp;
        DDRD |= 0b00010000;
        PORTD &= 0b11101111;
        _delay_us(2);
        DDRD &= 0b11101111;
        PORTD &= 0b11101111;
        _delay_us(10);

        if(PIND & 0b00010000) {         //if PD4 = 1 return 0 in temp = r24 and delay(49)
                temp = 1;

        }
        else{                   //else return 0 in temp = r24 and delay(49)
         temp = 0;
   }



   _delay_us(49);
   return temp;
}



void one_wire_transmit_bit(unsigned char x){

        unsigned char e;
        e = x;          // x will be r24 when the function is called
        DDRD |= 0b00010000;
        PORTD &= 0b11101111;
        _delay_us(2);

        if(e & 0x01){
                PORTD |= 0b00010000;
```

```
        }
        else{
                PORTD &= 0b11101111;
        }
        _delay_us(58);
        DDRD &= 0b11101111;
        PORTD &= 0b11101111;
        _delay_us(1);

}


unsigned char one_wire_receive_byte(){

    unsigned char temp;
        unsigned char temp2 = 0;       //temp2 = r26
        for(int i = 0; i < 8; i++){
                temp = one_wire_receive_bit();
                temp2 = temp2 >> 1;
                if(temp & 1) temp = 0x80;   //if r24 == 1,r24 = 0x80
                temp2 = temp2 | temp;              //r26 or r24
        }
        temp = temp2;
        return temp;
}



void one_wire_transmit_byte(unsigned char x){

    unsigned char temp;
        unsigned char temp2 = x;       //r26 = x = r24
        for(int i = 0; i < 8; i++){
                temp = 0;
                if(temp2 & 0x01) temp = 0x01;
                one_wire_transmit_bit(temp);
                temp2 = temp2 >> 1;

        }
}

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;
 //_delay_ms(1);
}

void usart_transmit_command(const uint8_t a[], int k){

    for(int i = 0; i< k; i++){
     // _delay_ms(50);
      usart_transmit(a[i]);
   //   _delay_us(10);
    }
    usart_transmit('\n');
}

void usart_transmit_word(const uint8_t a[], int k){

    for(int i = 0; i < k; i++){
     //   _delay_ms(50);
      usart_transmit(a[i]);
    }
}



void usart_transmit_payload(const uint8_t status[],int k){

    usart_transmit_word("ESP:payload:",12);


 /*   for(int i=0; i < sizeof(payload); i++){
        usart_transmit(payload[i]);
    } */


     usart_transmit('[');
        usart_transmit('{');
            usart_transmit("");
                usart_transmit_word("name", 4);
            usart_transmit("");

            usart_transmit(':');

            usart_transmit("");
                usart_transmit_word("temperature",11);
            usart_transmit("");

            usart_transmit(',');

            usart_transmit("");
                usart_transmit_word("value",5);
            usart_transmit("");

            usart_transmit(':');
```

```
    usart_transmit("");
    for(int j=0; j < 2; j++){
        usart_transmit(temp[j]);
    }
    usart_transmit("");
usart_transmit('}');


usart_transmit(',');


usart_transmit('{');

    usart_transmit("");
        usart_transmit_word("name",4);
    usart_transmit("");

    usart_transmit(':');

    usart_transmit("");
        usart_transmit_word("pressure",8);
    usart_transmit("");

    usart_transmit(',');

    usart_transmit("");
        usart_transmit_word("value",5);
    usart_transmit("");

    usart_transmit(':');


    usart_transmit("");
    for(int j=0; j < 2; j++){
        usart_transmit(pres[j]);
    }
    usart_transmit("");

usart_transmit('}');

usart_transmit(',');

usart_transmit('{');

    usart_transmit("");
        usart_transmit_word("name",4);
    usart_transmit("");

    usart_transmit(':');

    usart_transmit("");
        usart_transmit_word("team",4);
    usart_transmit("");
```

```
            usart_transmit(';');

            usart_transmit("");
               usart_transmit_word("value",5);
            usart_transmit("");

            usart_transmit(':');


            usart_transmit("");
               usart_transmit_word("39",2);
            usart_transmit("");

         usart_transmit('}');

         usart_transmit(',');

         usart_transmit('{');

            usart_transmit("");
               usart_transmit_word("name",4);
            usart_transmit("");

            usart_transmit(':');

            usart_transmit("");
               usart_transmit_word("status",6);
            usart_transmit("");

            usart_transmit(',');

            usart_transmit("");
               usart_transmit_word("value",5);
            usart_transmit("");

            usart_transmit(':');


            usart_transmit("");
            for(int j=0; j < k; j++){
               usart_transmit(status[j]);
            }
            usart_transmit("");
        usart_transmit('}');

 usart_transmit(']');

     usart_transmit('\n');
}


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

```
  // _delay_ms(1);
}




void metatroph(){
    z = ADCL;
    y = ADCH;
    z = z & 0b11000000;
    a1 = a2 = a3 = 0;

    while(1){
        if(y >= 51){
            a1 += 1;
            y -= 51;
        }
        else break;
    }

    z = z >> 6;
    y = y << 2;
    y = y | z;

    while(1){
        if(y >= 204){
            a1 += 1;
            y -= 204;
        }
        else break;
    }

    while(1){
        if(y >= 20){
            a2 += 1;
            y -= 20;
        }
        else break;
    }

    while(1){
        if(y >= 2){
            a3 += 1;
            y -= 2;
        }
        else break;
    }



    a3 = a3*4;
```

```
    a2 = a2*4;
    a1 = a1*4;
    a2 = a2 + (a3 / 10);
    a1 = a1 + (a2 / 10);
    a0 = a1/10;
    a3 = a3 - ((a3 /10)*10);
    a2 = a2 - ((a2 /10)*10);
    a1 = a1 - ((a1 /10)*10);

    p0 = a0;
    p1 = a1;
    p2 = a2;
    p3 = a3;

    a0 = a0 | 0b00110000;
    a1 = a1 | 0b00110000;
    a2 = a2 | 0b00110000;
    a3 = a3 | 0b00110000;

    pres[0] = a0;
    pres[1] = a1;




    return;
}



void get_temperature(){
    unsigned char  x;
    x = one_wire_reset();
    if(x == 0) {
        t1 = 0x00;
        t2 = 0x80;
     return;
    }
    one_wire_transmit_byte(0xCC);
    one_wire_transmit_byte(0x44);
    while(!(one_wire_receive_bit())) {
    }
    x = one_wire_reset();
    one_wire_transmit_byte(0xCC);
    one_wire_transmit_byte(0xBE);
    t1 = one_wire_receive_byte();
    t2 = one_wire_receive_byte();
//    t1 = 0xFF;
//    t2 = 0xFF;

}
```

```
void lcd_word(uint8_t word[],int k){
    for(int i=0; i < k; i++){
        lcd_data(word[i]);
    }
}

void lcd_received(){
    int z = 0;

    for(int k = 0; k<50; k++){         //Arxikopoihsh/ClearBuffer
         rec[k] = '\0';
     }

    for(int k = 0; k<50; k++){       //Receive
        rec[z] = usart_receive();
        if(rec[z] == '\n') break;
        z++;
     }
     z=0;

     for(int k = 0; k < 50; k++){   //Display
        if(rec[z] == '\n') break;
        lcd_data(rec[z]);
        z++;
     }

}

void usart_flash(void){
    unsigned char dump;
    int n = 0;
    while((UCSR0A&(1<<RXC0))&&(n<10)){
        dump=UDR0;
        n++;
    }
}

void buffer_clear(){
    usart_flash();

    for(int k = 0; k<50; k++){         //Arxikopoihsh/ClearBuffer
         rec[k] = '\0';
     }


}




/*
void usart_flash(){
    unsigned char dump;
```

```
    while (!(UCSR0A&(1<<TXC0))){
        dump=UDR0;
    }


} */


int main(){


   int flag = 0;
   uint8_t c;
      twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
    DDRC=0;
    ADMUX=0b01100000;
    ADCSRA=0b10000111;
    usart_init(103);
 while(1){

     DDRD |= 0xFF;

    usart_transmit_command("ESP:restart",11);
        //buffer_clear();
        lcd_init();
        //lcd_received();
        buffer_clear();

        _delay_ms(500);


        usart_transmit_command("ESP:connect",11);

        lcd_init();

        lcd_data('1');
        lcd_data('.');

        lcd_received();

        buffer_clear();
         _delay_ms(1000);



     usart_transmit_command("ESP:url:\"http://192.168.1.2:5050/data\"",38);



        lcd_init();
        lcd_data('2');
```

```
    lcd_data('.');
    _delay_ms(1);
    lcd_received();

    buffer_clear();



     _delay_ms(1000);

  get_temperature();

a4 = 0;
x1 = (uint16_t)t1;
x2 = (uint16_t)t2;
x2 = x2 << 8;
x1 = x1 + x2;

x1 = x1 >> 4;
x1 = x1 & 0x00FF;

a1 = 0;
a2 = 0;
a3 = 0;
    while(x1 >= 100){
        x1 = x1 -100;
        a1 += 1;
    }
    while(x1 >= 10){
        x1 = x1 - 10;
        a2 += 1;
    }
      while(x1 >= 1){
        x1 = x1 - 1;
        a3 += 1;
    }
    a3 = a3 + 5;
    a2 = a2 + 1 + (a3 / 10);
    a3 = a3 - ((a3 / 10)*10);

    te1= a1;
    te2 = a2;
    te3 = a3;

    a1 |= 0b00110000;
    a2 |= 0b00110000;
    a3 |= 0b00110000;

    temp[0] = a2;
    temp[1] = a3;

    DDRD |=  0b11111111;
    DDRB |=  0b11111111;
```

```
    DDRC |=  0b00000000;



       _delay_ms(2);

        ADCSRA |= (1 << ADSC);

       while(ADCSRA & 0b01000000){

       }
       metatroph();









    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_OUTPUT_1, 0xFD);
//PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT0 as output

    //s=4;
    for(int i = 0; i < 100; i++)
    {
      _delay_ms(10);
       y =  ~PCA9555_0_read(REG_INPUT_1);
       if(y & 0x40){
          flag = 1;
          s = 1;    break;       //NURSE CALL = 1

       }
    }
    s=4;
    PCA9555_0_write(REG_OUTPUT_1, 0xFE);
    for(int i = 0; i < 100; i++)
    {
       _delay_ms(10);
       y =  ~PCA9555_0_read(REG_INPUT_1);
       if(y & 0x40){
          flag = 0;
          s = 4;     break;      //OK = 4

       }
    }

    te3 = te3 - 1;
    if((te2 == 3 && te3 > 7) || (te2 < 3) || (te2 == 3 && te3 < 4) || (te2 > 3) ){
       s = 2;  //CHECK TEMP
    }
```

```
else if((p0 > 0 && p1 > 2) || (p0 == 0 && p1 < 4) ){
    s = 3;  //CHECK PRESSURE
}


lcd_init();
lcd_data(temp[0]);
lcd_data(temp[1]);
lcd_data(' ');
lcd_data(pres[0]);
lcd_data(pres[1]);
lcd_command(0b11000000);

if(flag == 1){                          //NURSE CALL = 1

     lcd_word(status1,9);
     _delay_ms(1000);
     usart_transmit_payload(status1,9);

}
else if(s == 2 && flag == 0){                       //CHECK TEMP
     lcd_word(status2,9);
     _delay_ms(1000);
     usart_transmit_payload(status2,9);

}
else if(s == 3 && flag == 0){                       //CHECK PRESSURE
     lcd_word(status3,13);
     _delay_ms(1000);
     usart_transmit_payload(status3,13);

}
else if(s == 4 && flag == 0){
    lcd_word(status4,2);                        //OK = 4
    _delay_ms(1000);
    usart_transmit_payload(status4,2);

}

lcd_init();
lcd_data('3');
lcd_data('.');
lcd_received();
buffer_clear();

_delay_ms(1000);

usart_transmit_command("ESP:transmit",12);

lcd_init();
lcd_data('4');
lcd_data('.');
lcd_received();
    buffer_clear();
```

```
        _delay_ms(1000);

    }
}
```