Συνδυαστική/Επαναληπτική άσκηση – Εφαρμογή Internet of Things

Εργαστήριο Μικροϋπολογιστών

Κριθαρίδης Κωνσταντίνος, *el*21045 Μπαλάτος Δημήτριος, *el*21170

10 Δεκεμβρίου 2024

1 Ζήτημα 8.1

Στα ζητήματα αυτής της εργαστηριακής άσκησης, επειδή απαιτούταν η χρήση πολλών διαφορετικών συσκευών, αποφασίσαμε για καλύτερη διάρθρωση και τμηματοποίηση του κώδικα να χρησιμοποιήσουμε διαφορετικά header files, με τις μεθόδους υπεύθυνες για κάθε συσκευή.

Για το συγκεκριμένο ζήτημα, δημιουργήσαμε αρχικά το header file `usart.h` στο οποίο εισαγάγαμε τον κώδικα που δινόταν από την εκφώνηση για αρχικοποίηση, αποστολή και λήψη ενός byte, τον οποίο επεκτείναμε με τις μεθόδους που κρίναμε απαραίτητες. Πιο συγκεκριμένα, γράψαμε τις `usart_transmit_string` (στέλνει ένα string μέσω της UART), `usart_receive_string` (λαμβάνει ένα string από τη UART), `create_command` (φτιάχνει string που αντιστοιχεί στο format του ορισμού μεταβλητής στο payload), `create_payload` (φτιάχνει string για την εντολή payload), `usart_command` (στέλνει εντολή στο UART και λαμβάνει την απάντηση), `usart_restart` (κάνει restart το ESP8266) και `usart_connect` (πραγματοποιεί τη σύνδεση στο WiFi). Ο κώδικας του header file `usart.h` παρατίθεται στο τέλος, μαζί με όλα τα υπόλοιπα header files που χρησιμοποιήθηκαν για τα ζητήματα της εργαστηριακής άσκησης.

Επιπλέον, χρησιμοποιήσαμε την οθόνη LCD μέσω του Port Expander για να εμφανίζουμε τα ζητούμενα μηνύματα στην οθόνη, οπότε χρησιμοποιήσαμε και τον κώδικα που είχα γράψει σε προηγούμενες εργαστηριακές ασκήσεις για αυτά. Τα header files `pca9555.h` για το Port Expander και το `lcd_pex.h` που το χρησιμοποιεί για την οθόνη LCD παρατίθενται και αυτά στο τέλος της αναφοράς.

Στον πηγαίο κώδικα που ακολουθεί, αφού αρχικοποιήσουμε το USART, το PCA9555 και την οθόνη LCD, συνδεόμαστε στο WiFi (τυπώνοντας ανάλογα με την απάντηση που λαμβάνουμε το αντίστοιχο μήνυμα στην οθόνη), και, αφού περιμένουμε λίγο για να παρατηρήσουμε την απάντηση, ρυθμίζουμε το URL στο οποίο θα στέλνουμε δεδομένα (εμφανίζοντας πάλι ανάλογο μήνυμα στην οθόνη).

```c
/*
 * main.c
 *
 * Created: 12/6/2024 11:16:24 AM
 *  Author: User
 */

#include "../../libs/usart.h"
#include "../../libs/lcd_pex.h"

```

```c
int main(void)
{
        int ret;
        char buf[30];

        usart_init(UBRR);
        twi_init();
        PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output for lcd display
        lcd_init();

        // usart_restart();
        ret = usart_connect();
        if (ret)
                snprintf(buf, sizeof(buf), "1.Fail (%d)", ret);
        else
                snprintf(buf, sizeof(buf), "1.Success");
        lcd_string(buf);

        _delay_ms(5000);

        if (usart_command("ESP:url:\"http://192.168.1.250:5000/data\""))
                snprintf(buf, sizeof(buf), "2.Fail (%d)", ret);
        else
                snprintf(buf, sizeof(buf), "2.Success");
        lcd_string(buf);

        while (1);

}
```

2 Ζήτημα 8.2

Στο ζήτημα αυτό χρειάστηκε ακόμα επικοινωνία one-wire με τον αισθητήρα θερμοκρασίας DS18B20 (`ds18b20.h`), χρήση πληκτρολογίου (`keypad.h`) και χρήση του ποτενσιόμετρου POT0 με ADC (`pot.h`). Και αυτά τα header files παρατίθενται στο τέλος της αναφοράς.

Στον πηγαίο κώδικα, αφού αρχικοποιήσουμε το one-wire και το ADC0 για το ποτενσιόμετρο, επεκτείνοντας τον πηγαίο κώδικα του προηγούμενου ζητήματος, διαβάζουμε διαρκώς μετρήσεις θερμοκρασίας από το DS18B20, πίεσης από το POT0 (ρυθμίζοντας κατάλληλα τις μετρήσεις και των δύο) και το πληκτρολόγιο για εντολές. Αν πατηθεί το πλήκτρο '8', που αντιστοιχεί στο τελευταίο ψηφίο της ομάδας μας (28), το status γίνεται `NURSE CALL` . Αλλιώς, αν η πίεση είναι πάνω από 12 ή κάτω από 4 γίνεται `CHECK PRESSURE` , ενώ αν η θερμοκρασία είναι κάτω από 34 ή πάνω από 37 γίνεται `CHECK TEMP` . Αλλιώς, το status γίνεται `OK` . Το status είναι σε μορφή έτοιμη ώστε με τις `create_command` και `create_payload` να ενσωματωθεί στο payload. Τέλος, εμφανίζουμε στην 1η γραμμή της οθόνης LCD τις μετρήσεις θερμοκρασίας και πίεσης, και στην 2η γραμμή το status. Ο πηγαίος κώδικας αυτός αυτό φαίνεται παρακάτω.

```c
/*
 * main.c
 *
```

```
 * Created: 12/6/2024 11:16:24 AM
 *  Author: User
 */

#include "../../libs/usart.h"
#include "../../libs/ds18b20.h"
#include "../../libs/lcd_pex.h"
#include "../../libs/keypad.h"
#include "../../libs/pot.h"

int main(void)
{
        int ret;
        char buf[30];

        usart_init(UBRR);
        twi_init();
        PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output for
 ↪  lcd display
        lcd_init();
    one_wire_reset();
        pot_init();

        // usart_restart();
        ret = usart_connect();
        if (ret)
                snprintf(buf, sizeof(buf), "1.Fail (%d)", ret);
        else
                snprintf(buf, sizeof(buf), "1.Success");
        lcd_string(buf);

        _delay_ms(5000);

        if (usart_command("ESP:url:\"http://192.168.1.250:5000/data\""))
                snprintf(buf, sizeof(buf), "2.Fail (%d)", ret);
        else
                snprintf(buf, sizeof(buf), "2.Success");
        lcd_string(buf);

    _delay_ms(5000);

    int16_t raw_temp;
    int8_t pressure;
    char keypad;
    char status[16];
    char buf[60];

    while (1) {
        raw_temp  = read_temp();
        if (raw_temp == TEMP_ERR) raw_temp = 0;
        add_to_temp(&raw_temp, 10);
```

```
        pressure = read_pressure();
        keypad = keypad_to_ascii();

        if (keypad == '8')
            sprintf(status, "NURSE CALL");
        else if (keypad == '#' && !strcmp(status, "NURSE CALL"))
            sprintf(status, "OK");
        else if (pressure < 4 || 12 < pressure)
            sprintf(status, "CHECK PRESSURE");
        else if (TEMP(raw_temp) < 34 || 37 < TEMP(raw_temp))
            sprintf(status, "CHECK TEMP");
        else if (strcmp(status, "NURSE CALL"))
                        sprintf(status, "OK");

        char* pressure_str = pressure_to_str(pressure);
        char* temp_str = temp_to_str(raw_temp, 1);

                strcpy(buf, "T: ");
                strcpy(buf + strlen(buf), temp_str);
                strcpy(buf + strlen(buf), " P: ");
                strcpy(buf + strlen(buf), pressure_str);
                strcpy(buf + strlen(buf), "\n");
                strcpy(buf + strlen(buf), status);

                lcd_string(buf);
                _delay_ms(1000);
    }

}
```

3 Ζήτημα 8.3

Στο ζήτημα αυτό χρειάστηκε ακόμα επικοινωνία μέσω UART με τον server για αποστολή πληροφοριών (θερμοκρασία, πίεση, status). Για να γίνει αυτό, κατασκευάζουμε το JSON αντικείμενο για αποστολή μέσω των εντολών `payload` – `transmit` . Για να απλοποιήσουμε την διαδικασία, ορίσαμε 2 συναρτήσεις:

- `create_command(key, value)`: κατασκευάζει ένα JSON ζευγάρι key-value (σε string)
- `create_payload(argc, argv)`: ενώνει τα ζευγάρια σε ένα JSON αντικείμενο και προσθέτει το πρόθεμα `"ESP:payload"`.

Για αποφυγή προβλημάτων, επανασυνδεόμαστε στον server μέσω UART, ετοιμάζουμε το πακέτο μέσω εντολής `payload` και μετά το στέλνουμε μέσω της `transmit`

```
/*
 * main.c
 *
 * Created: 12/6/2024 11:16:24 AM
 *  Author: User
 */

#define PRECISION 4 // binary precision bits
```

```
9
10  #include "../../libs/usart.h"
11  #include "../../libs/ds18b20.h"
12  #include "../../libs/lcd_pex.h"
13  #include "../../libs/keypad.h"
14  #include "../../libs/pot.h"
15
16  int main(void)
17  {
18          int ret;
19          char buf[60];
20
21          usart_init(UBRR);
22          twi_init();
23          PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output for
     ↪  lcd display
24          lcd_init();
25          one_wire_reset();
26          pot_init();
27
28
29      int16_t raw_temp;
30      int8_t pressure;
31      char keypad;
32      char status[16];
33
34      while (1) {
35          raw_temp  = read_temp();
36                  if (raw_temp == TEMP_ERR) raw_temp = 0;
37          add_to_temp(&raw_temp, 10);
38          pressure = read_pressure();
39          keypad = keypad_to_ascii();
40
41          if (keypad == '8')
42              sprintf(status, "NURSE CALL");
43          else if (keypad == '#' && !strcmp(status, "NURSE CALL"))
44              sprintf(status, "OK");
45          else if (pressure < 4 || 12 < pressure)
46              sprintf(status, "CHECK PRESSURE");
47          else if (TEMP(raw_temp) < 34 || 37 < TEMP(raw_temp))
48              sprintf(status, "CHECK TEMP");
49                  else if (strcmp(status, "NURSE CALL"))
50                          sprintf(status, "OK");
51
52          char* pressure_str = pressure_to_str(pressure);
53          char* temp_str = temp_to_str(raw_temp, 1);
54
55                  strcpy(buf, "T: ");
56                  strcpy(buf + strlen(buf), temp_str);
57                  strcpy(buf + strlen(buf), " P: ");
58                  strcpy(buf + strlen(buf), pressure_str);
```

```
                strcpy(buf + strlen(buf), "\n");
                strcpy(buf + strlen(buf), status);

                lcd_string(buf);
                _delay_ms(1000);

                char* json[4] = {
                        create_command("temperature", temp_str),
 ↪ create_command("pressure", pressure_str),
                        create_command("team", "28")           ,
 ↪ create_command("status", status)
                };
                lcd_string(json[0]);
                _delay_ms(1000);
                char* cmd = create_payload(4, json);
                lcd_string(cmd);
                _delay_ms(2000);

                //usart_restart();
                ret = usart_connect();
                if (ret)
                snprintf(buf, 60, "1.Fail (%d)", ret);
                else
                snprintf(buf, 60, "1.Success");
                lcd_string(buf);

                _delay_ms(1000);

                lcd_string("Moving on");
                if
                 ↪  (usart_command("ESP:url:\"http://192.168.1.250:5000/data\"\n"))
                snprintf(buf, 60, "2.Fail (%d)", ret);
                else
                snprintf(buf, 60, "2.Success");
                lcd_string(buf);

                _delay_ms(1000);

                usart_command(cmd);
                usart_command("ESP:transmit\n");
//
        free(pressure_str);
        free(temp_str);
        free(cmd);
                for (int i=0; i<4; ++i) free(json[i]);

        _delay_ms(1000);
    }

}
```

4 Header files

```
#ifndef __UTILS_H__
#define __UTILS_H__

#include <xc.h>

#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#include <inttypes.h>
#include <stdbool.h>

#endif // __UTILS_H__
```

```
#ifndef __USART_H__
#define __USART_H__

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const int BAUD = 9600;
const int UBRR = 103;

/* Routine: usart_init
Description: This routine initializes the usart as shown below.
------- INITIALIZATIONS -------
Baud rate: 9600 (Fck= 8MH)
Asynchronous mode
Transmitter on
Receiver 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;
}

void usart_transmit_string(const char *str){
  while(*str != '\0'){
    usart_transmit(*str++);
  }
}

char* usart_receive_string(){
  char *str = (char*) malloc(1024*sizeof(char));
  int i = 0;
  while((str[i++] = usart_receive()) != '\n');
  str[i] = '\0';
  return str;
}

char* create_command(const char *name, const char* value){
  char *buff = (char*) malloc(64*sizeof(char));
  strcpy(buff, "{\"name\":\"");
  strcpy(buff + strlen(buff), name);
  strcpy(buff + strlen(buff), "\",\"value\":\"");
  strcpy(buff + strlen(buff), value);
  strcpy(buff + strlen(buff), "\"}");
  return buff;
}

char* create_payload(int argc, char **argv){
  char *buff = (char*) malloc(312*sizeof(char));
  strcpy(buff, "ESP:payload:[");
  for(int i = 0; i < argc; i++){
          strcpy(buff + strlen(buff), argv[i]);
          strcpy(buff + strlen(buff), (i == argc-1) ? "]\n" : ",");
  }
  return buff;
}

static enum { NO_RESTART, RESTART } usart_state = NO_RESTART;

static int are_same (const char *a, const char *b)
{
        while (*a != '\0' && *b != '\0')
```

```
                if (*a != *b) return 1;
                else a++, b++;
        return (*a != '\0') || (*a != '\0');
}

int usart_command(const char *cmd)
{
        char *buf = NULL;
        int ret;


        _delay_ms(500);
        usart_transmit_string(cmd);
        buf = usart_receive_string();
        ret = are_same(buf, "\"Success\"\n");

        free(buf);
        return ret;
}

void usart_restart()
{
        char *buf;
        usart_transmit_string("ESP:restart\n");
        buf = usart_receive_string(); // restart response
        free(buf);

        buf = usart_receive_string(); // restart response
        free(buf);

        usart_state = RESTART;
}

int usart_connect()
{
        //int tmp = -1;
                //goto connect;
//
        //tmp = 1;
        //if (usart_command("ESP:ssid:\"Micro_IoT\"\n"))
                //return 1;
        //if (usart_command("ESP:password:\"Microlab_IoT\"\n"))
                //return 2;
        //if (usart_command("ESP:debug:\"false\"\n"))
                //return 3;
        //if (usart_command("ESP:baudrate:\"9600\"\n"))
                //return 4;

        if (usart_command("ESP:connect\n"))
                return 5;
        return 0;
```

```
}

#endif // __USART_H__
```

```
#ifndef __POT_H__
#define __POT_H__

void pot_init(){
        // Fast PWM, 8 bit, non-inverting output, N = 256. BOTTOM = 0, TOP =
        ↪  0x00ff = 255
        TCCR1A = (1<<WGM10) | (1<<COM1A1);
        TCCR1B = (1<<WGM12) | (1<<CS12);

        // Init ADC:
        //   Vref = 5V, ADC0
        ADMUX = (1 << REFS0);
        //   Enable, no interrupt, no conversion, 125 kHz
        ADCSRA = (1 << ADEN) | (0 << ADSC) | (0 << ADIE) | (7 << ADPS0);
}

int16_t read_pot(){
        // Handle ADC
        ADCSRA |= (1 << ADSC);
        while (ADCSRA & (1 << ADSC));
        return ADC & ((1 << 10) - 1);
}

int8_t read_pressure(){
        int16_t pressure = read_pot();
        return pressure/(1<<10)*20;
}

char* pressure_to_str(int8_t pressure){
        char *buff = (char*) malloc(1024*sizeof(char));
        char num[5];
        int pos = 0, idx = 0;

        if (pressure == 0) {
                strcpy(buff, "0");
        } else {
                for (; pressure; pressure /= 10, pos++)
                        num[pos] = (pressure % 10) + '0';
                for (pos -= 1; pos >= 0; pos--)
                        buff[idx++] = num[pos];
                buff[idx] = '\0';
        }
        return buff;
}

#endif //__POT_H__
```

```
#ifndef __PCA_H__
#define __PCA_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

#define NOP() do { __asm__ __volatile__ ( "nop "); } while (0)

// 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)));
        // 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));
}

uint8_t LAST;

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();
        LAST = value;
        //if (reg != REG_CONFIGURATION_0) exit(0);
}

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;
}

#endif // __PCA_H__
```

```
#ifndef __LCD_H__
#define __LCD_H__

#include "pca9555.h"

void flash ()
{
        _delay_us(50);
        uint8_t tmp = PCA9555_0_read(REG_INPUT_0);
        PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 3));
```

```
        _delay_us(50);
        PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 3));
}

void write_2_nibbles(uint8_t data){
        uint8_t temp = LAST & 0x0f;
        uint8_t out = data & 0xf0 | temp;
        PCA9555_0_write(REG_OUTPUT_0, out);
        flash();

        out = (data << 4) & 0xf0 | temp;
        PCA9555_0_write(REG_OUTPUT_0, out);
        flash();
}

void lcd_data (uint8_t data)
{
        uint8_t tmp = LAST;
        PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 2));
        write_2_nibbles(data);
        _delay_us(500);
}

void lcd_command (uint8_t instr)
{
        uint8_t tmp = LAST;
        PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 2));
        write_2_nibbles(instr);
        _delay_us(500);
}

void lcd_clear_display(){
        lcd_command(0x01);
        _delay_ms(200);
}

void lcd_init ()
{
        _delay_ms(200);

        uint8_t out = 0x30;
        for (int i=0; i<3; ++i) {
                PCA9555_0_write(REG_OUTPUT_0, out);
                flash();
                _delay_us(250);
        }
        PCA9555_0_write(REG_OUTPUT_0, 0x20);
        flash();
        _delay_us(250);

        lcd_command(0x28);
```

```
62          lcd_command(0x0c);
63          lcd_clear_display();
64          lcd_command(0x06);
65  }
66
67  void lcd_string (const char* str)
68  {
69          lcd_clear_display();
70          for (; *str != '\0'; str++) {
71                  if (*str == '\n')
72                          lcd_command(0xc0);
73                  else
74                          lcd_data(*str);
75          }
76  }
77
78  void lcd_temp (int16_t val, int decimals)
79  {
80          char tmp[17];
81          int idx = 0;
82          lcd_command(0x02);
83
84          if (val < 0) {
85                  lcd_data('-');
86                  val = -val;
87          }
88
89          int   int_part  = val >> PRECISION;
90          float frac_part =  ((float)(val) / (1 << PRECISION)) - int_part;
91
92          if (int_part) {
93                  for (; int_part; int_part /= 10)
94                          tmp[idx++] = (int_part % 10) + '0';
95                  for (idx -= 1; idx >= 0; --idx)
96                          lcd_data(tmp[idx]);
97          } else {
98                  lcd_data('0');
99          }
100
101         lcd_data('.');
102         for (int i=0; i<decimals; ++i) {
103                 frac_part *= 10;
104                 lcd_data((int)(frac_part) + '0');
105                 frac_part -= (int)(frac_part);
106         }
107
108         lcd_data(0b11011111); lcd_data('c');
109 }
110
111 #endif // _LCD_H__
```

```
#ifndef __PCA_H__
#define __PCA_H__

#include "pca9555.h"

uint8_t scan_row(uint8_t row){ //row = 0, 1, 2, 3
        uint8_t mask = 0x0f & ~(1<<row);
        PCA9555_0_write(REG_OUTPUT_1, mask); //enable row as input
        _delay_us(100);
        uint8_t in = ~PCA9555_0_read(REG_INPUT_1); //read columns of row pressed
         ↪  in positive logic
        in >>= 4; //remove IO1[0:3]
        return in; //4 bits
}

uint16_t scan_keypad(){
        uint16_t row0 = scan_row(0);
        uint16_t row1 = scan_row(1);
        uint16_t row2 = scan_row(2);
        uint16_t row3 = scan_row(3);
        return row0 | (row1<<4) | (row2<<8) | (row3<<12);
}

uint16_t scan_keypad_rising_edge(){
        static uint16_t pressed_keys = 0;
        uint16_t pressed_keys_tempo = scan_keypad();
        _delay_ms(15); //wait to avoid triggering
        pressed_keys_tempo &= scan_keypad(); //only keep the actual buttons
 ↪  pressed
        uint16_t keys_just_pressed = pressed_keys_tempo & (~pressed_keys);
        pressed_keys = pressed_keys_tempo;
        return keys_just_pressed;
}

char keypad_to_ascii(){
        uint16_t key = scan_keypad_rising_edge();
        if(key&(1<<0)) return '*';
        if(key&(1<<1)) return '0';
        if(key&(1<<2)) return '#';
        if(key&(1<<3)) return 'D';
        if(key&(1<<4)) return '7';
        if(key&(1<<5)) return '8';
        if(key&(1<<6)) return '9';
        if(key&(1<<7)) return 'C';
        if(key&(1<<8)) return '4';
        if(key&(1<<9)) return '5';
        if(key&(1<<10)) return '6';
        if(key&(1<<11)) return 'B';
        if(key&(1<<12)) return '1';
        if(key&(1<<13)) return '2';
        if(key&(1<<14)) return '3';
```

```
        if(key&(1<<15)) return 'A';
        return 0;
}

#endif //__PCA_H__
```

```
#ifndef _DS18B20_H_
#define _DS18B20_H_

#include "utils.h"

#define SET(x, b)   do { (x) |=   1 << (b);  } while (0)
#define CLEAR(x, b) do { (x) &= ~(1 << (b)); } while (0)

/* TEMPLATE CODE FROM GIVEN ASSEMBLY */
/* {{{ */
static bool one_wire_reset (void)
{
        SET(DDRD, 4); // set output

        CLEAR(PORTD, 4); // sent 0
        _delay_us(480);

        CLEAR(DDRD, 4); // set input
        CLEAR(PORTD, 4); // disable pull-up
        _delay_us(100);

        int tmp = PIND; // read PORTD
        _delay_us(380);
        return (tmp & (1 << 4)) == 0;
}

static uint8_t one_wire_receive_bit (void)
{
        SET(DDRD, 4); // Set output
        CLEAR(PORTD, 4);
        _delay_us(2);

        CLEAR(DDRD, 4); // set input
        CLEAR(PORTD, 4); // disable pull-up
        _delay_us(10);

        uint8_t ret = PIND & (1 << 4);
        _delay_us(49);
        return ret >> 4;
}

static void one_wire_transmit_bit (uint8_t in)
{
        SET(DDRD, 4); // Set output
        CLEAR(PORTD, 4);
```

```
        _delay_us(2);

        in &= 1; // keep only LSB
        if (in) SET(PORTD, 4);
        else    CLEAR(PORTD, 4);

        _delay_us(58);
        CLEAR(DDRD, 4); // set input
        CLEAR(PORTD, 4); // disable pull-up
        _delay_us(1);
}

static uint8_t one_wire_receive_byte (void)
{
        uint8_t ret = 0;
        for (int i=0; i<8; ++i)
                ret |= (one_wire_receive_bit() << i);
        return ret;
}

static void one_wire_transmit_byte (uint8_t in)
{
        for (int i=0; i<8; ++i, in >>= 1)
                one_wire_transmit_bit(in);
}
/* }}} */

#define TEMP_ERR 0x8000

int16_t read_temp (void)
{
        int16_t ret = 0;

        if (one_wire_reset() != 1) return TEMP_ERR;
        one_wire_transmit_byte(0xCC);
        one_wire_transmit_byte(0x44);
        while (one_wire_receive_bit() != 1);

        if (one_wire_reset() != 1) return TEMP_ERR;
        one_wire_transmit_byte(0xCC);
        one_wire_transmit_byte(0xBE);

        ret |= one_wire_receive_byte();
        ret |= one_wire_receive_byte() << 8;
        return ret;
}

#define TEMP(raw) ((raw) >> PRECISION)

void add_to_temp(int16_t *temp, int8_t add)
{
```

```
97      *temp += (add << PRECISION);
98  }
99
100 char *temp_to_str(int16_t val, int decimals)
101 {
102         char *temp_buf = malloc(64);
103         char tmp[17];
104         int idx = 0, pos = 0;
105
106         if (val < 0) {
107                 temp_buf[pos++] = '-';
108                 val = -val;
109         }
110
111         int   int_part  = val >> PRECISION;
112         float frac_part =  ((float)(val) / (1 << PRECISION)) - int_part;
113
114         if (int_part) {
115                 for (; int_part; int_part /= 10)
116                         tmp[idx++] = (int_part % 10) + '0';
117                 for (idx -= 1; idx >= 0; --idx)
118                         temp_buf[pos++] = tmp[idx];
119         } else {
120                 temp_buf[pos++] = '0';
121         }
122
123         temp_buf[pos++] = '.';
124         for (int i=0; i<decimals; ++i) {
125                 frac_part *= 10;
126                 temp_buf[pos++] = (int)(frac_part) + '0';
127                 frac_part -= (int)(frac_part);
128         }
129
130         temp_buf[pos++] = 'C';
131         temp_buf[pos] = '\0';
132         return temp_buf;
133 }
134
135 #endif // _DS18B20_H_
```