Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΚΑΔ. ΕΤΟΣ 2024-2025

7ή εργαστηριακή άσκηση.

Ομάδα : 58

Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076

Ονοματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Στην συγκεκριμένη Εργαστηριακή Άσκηση θα χρησιμοποιήσουμε σειριακή Επικοινωνία 1-wire για χρήση αισθητήρα θερμοκρασίας. Να σημειωθεί πως χρησιμοποιούμε αισθητήρα DS18B20 και όχι DS1820.

```
#define F_CPU 16000000UL
#include "twi_pca.h"
#include "avr/io.h"
#include <util/delay.h>
#include <stdbool.h>
#include <stdint.h>
#include <avr/interrupt.h>

uint8_t fake_d = 0x00; //global to simulate PORTD

//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)
//TWEA =1 -> ACK
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
//TWEA =0 -> No ACK
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;          // Isn't the & unnecessary
	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(uint8_t input) {
	unsigned char temp = input & 0xF0;
	unsigned char pind  = fake_d & 0x0F;
	fake_d = pind + temp;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	fake_d |= 0x08;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	asm("nop");
	asm("nop");

	fake_d &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_0, fake_d);


	temp = input & 0x0F;
	temp = temp << 4;

	fake_d = pind + temp;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	fake_d |= 0x08;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	asm("nop");
	asm("nop");

	fake_d &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	return;
}

void lcd_data (uint8_t input) {

	fake_d |= 0x04;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	write_2_nibbles(input);
	_delay_ms(1);
	return;
}

void lcd_command (uint8_t input) {

	fake_d &= ~(0x04);
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	write_2_nibbles(input);
	_delay_ms(1);
	return;
```

```
}

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

void lcd_change_line () {
	//I send command 0xC0 so I write in DDRAM Address the 100 0000
	lcd_command(0xC0);
	_delay_ms(5);
	return;
}

void lcd_init () {
	_delay_ms(200);

	fake_d = 0x30;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	for (int i=0; i<3; i++) {

		fake_d |= 0x08;
		PCA9555_0_write(REG_OUTPUT_0, fake_d);

		asm("nop");
		asm("nop");

		fake_d &= ~(0x08);
		PCA9555_0_write(REG_OUTPUT_0, fake_d);

		_delay_ms(1);
	}

	fake_d = 0x20;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	fake_d |= 0x08;
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	asm("nop");
	asm("nop");

	fake_d &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_0, fake_d);

	_delay_ms(1);

	lcd_command(0x28);
	lcd_command(0x0c);
	lcd_clear_display();
	lcd_command(0x06);
	return;
}
```

```
#ifndef TWI_PCA_HEADER
#define      TWI_PCA_HEADER

#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)
//MY COMMENT: Here pre-scalar is taken as 1 - it will be set so in twi_init
#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

// For reading only TWS[7:3] from TWSR0
#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)

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

void twi_init(void);
unsigned char twi_readAck(void);
unsigned char twi_readNak(void);
unsigned char twi_start(unsigned char address);
void twi_start_wait(unsigned char address);
unsigned char twi_write( unsigned char data );
```

```
unsigned char twi_rep_start(unsigned char address);
void twi_stop(void);

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value);
uint8_t PCA9555_0_read(PCA9555_REGISTERS reg);

void write_2_nibbles (uint8_t input);
void lcd_data (uint8_t input);
void lcd_command (uint8_t input);
void lcd_clear_display ();
void lcd_change_line ();
void lcd_init ();


#endif /* TWI_PCA_HEADER */
```

Στην συνέχεια μεταφράζουμε τις ρουτίνες Assembly που μας δίνονται από το εργαστήριο σε C, ώστε να μπορούμε να χειριζόμαστε την σειριακή επικοινωνία 1-wire. Ο κώδικας αυτών βρίσκεται παρακάτω:

```
bool one_wire_reset() {
	uint8_t temp;

	//Set PD4 as output and equal to 0
	DDRD |= 0x10;
	PORTD &= ~(0x10);

	_delay_us(480);   //480 uSec Reset Pulse

	//Set PD4 as input and disable pull-up
	DDRD &= ~(0x10);
	PORTD &= ~(0x10);

	//Wait 100 usec for connected devices to transmit the presence pulse
	_delay_us(100);

	//temp = PD4
	temp = PIND & 0x10;
	_delay_us(380);

	//If a connected device is detected(PD4==0) return 1 else return 0
	if(temp == 0)
	{
		return 1;
	}
	else
	{
		return 0;
	}
}

uint8_t one_wire_receive_bit() {
	uint8_t result = 0;
```

```
	//Set PD4 as output and equal to 0
	DDRD |= 0x10;
	PORTD &= ~(0x10);
	_delay_us(2);

	//Set PD4 as input and disable pull-up
	DDRD &= ~(0x10);
	PORTD &= ~(0x10);
	_delay_us(10);

	//Store PD4 in LSB of result
	result = (PIND & 0x10) >> 4;

	//Delay 49 uSec to meet the standards
	_delay_us(49);

	return result;
}

void one_wire_transmit_bit(uint8_t input) {
	bool one = input & 0x01;

	//Set PD4 as output and equal to 0
	DDRD |= 0x10;
	PORTD &= ~(0x10);
	_delay_us(2);

	//PD4 = LSB of input
	if(one)
	{
		PORTD |= 0x10;
	}
	else
	{
		PORTD &= ~(0x10);
	}
	_delay_us(58);

	//Set PD4 as input and disable pull-up
	DDRD &= ~(0x10);
	PORTD &= ~(0x10);
	_delay_us(1);
	return;
}


uint8_t one_wire_receive_byte() {
	uint8_t result =0;
	//Load in result from LSB to MSB
	for(int i=0; i<8; i++)
	{
		result |= (one_wire_receive_bit() << i);
	}
	return result;
```

```
}
void one_wire_transmit_byte(uint8_t input) {
	uint8_t temp;
	//We transmit from LSB to MSB
	for(int i=0; i<8; i++)
	{
		temp = input & (1<<i);
		temp = temp >> i;
		one_wire_transmit_bit(temp);
	}
	return;
}
```

Ακόμα, υλοποιούμε την συνάρτηση που μας ζητείται και την ονομάζουμε our_func(), με comments που εξηγούν τις λειτουργίες που επιτελεί:

```
uint16_t our_func() {
	bool flag = 0; // If flag = 1 then we haven't found a device
	if(!one_wire_reset())
	{
		flag = 1;
	}
	one_wire_transmit_byte(0xCC);		// Send command to bypass the selection of
device, since we only have 1 device
	one_wire_transmit_byte(0x44);		// Send command to start the measurement
of temperature
	while(one_wire_receive_bit() != 1); // Loop up until the device sends bit = 1
	if(!one_wire_reset())
	{
		flag = 1;
	}
	one_wire_transmit_byte(0xCC);		// Send command to bypass the selection of
device, since we only have 1 device
	one_wire_transmit_byte(0xBE);		// Send command to read 16-bit temperature

	uint16_t reg24, reg25;
	reg24 = one_wire_receive_byte();
	reg25 = (one_wire_receive_byte()) << 8;
	reg25 |= reg24;
	if(flag)
	{
		return 0x8000;
	}
	else
	{
		return reg25;
	}
}
```

Τέλος, γράφουμε την main συνάρτηση η οποία διαβάζει συνεχόμενα data από την σειριακή επικοινωνία και αν δεν υπάρχει συνδεδεμένη συσκευή τυπώνει “NO Device” στην οθόνη, ενώ αν βρει σωστά τον αισθητήρα τυπώνει στην οθόνη (με την βοήθεια 3 ακόμα βοηθητικών συναρτήσεων) την θερμοκρασία σε δεκαδική τιμή 3 ψηφίων με πρόσημο. Ο κώδικας είναι ο παρακάτω:

```
uint8_t integer_part(uint16_t input) {
	uint8_t low, high, integer;
	low = input & 0x00FF;
	high = (input & 0xFF00) >> 8;
	integer = low >> 4;
	integer |= ((high << 4) & 0x70);
	return integer;
}

// Return the decimal part of our reading
int decimal_part(uint16_t input) {
	uint8_t low = input & 0x000F;
	int result = 0;
	if(low & 0x08)
	{
		result += 5000;
	}
	if(low & 0x04)
	{
		result += 2500;
	}
	if(low & 0x02)
	{
		result += 1250;
	}
	if(low & 0x01)
	{
		result += 625;
	}
	return result;
}

void send_three_to_screen(int input) {
	uint8_t to_send;
	to_send = input % 1000;
	to_send = input / 100 ;
	lcd_data('0'+to_send);
	to_send = input % 100;
	to_send = to_send / 10;
	lcd_data('0'+to_send);
	to_send = input % 10;
	lcd_data ('0'+to_send);
}
```

```
int main() {
	//For the LCD Screen
	twi_init();
	PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
	lcd_init();
	_delay_ms(100);			//Maybe for the LCD
	lcd_clear_display();

	DDRB = 0xFF;
	PORTB = 0x00;

	char problem[] = "NO Device";
	uint16_t reading;
	uint8_t int_part, dec_part;
	uint16_t previous_reading;
	while(1)
	{
		//Take reading
		reading = our_func();
		//If we have found no device
		if(reading == 0x8000)
		{
			lcd_clear_display();

			for(int i=0; i<9; i++)
			{
				lcd_data(problem[i]);
			}
			_delay_ms(100);
		}
		else
		{
			//If the reading hasn't changed go to the next iteration of the
while loop
			if(reading == previous_reading)
			{
				continue;
			}
			previous_reading = reading;
			lcd_clear_display();

			if((reading & 0x8000) == 1)	//Negative
			{
				reading = (~(reading)) + 1;
				lcd_data('-');
				int_part = integer_part(reading);
				dec_part = decimal_part(reading);
				send_three_to_screen(int_part);
				lcd_data('.');
				send_three_to_screen(dec_part);
				lcd_data(' ');
				lcd_data('o');
				lcd_data('C');
			}
			else				//Positive
```

```
                {
                        int_part = integer_part(reading);
                        dec_part = decimal_part(reading);
                        lcd_data('+');
                        send_three_to_screen(int_part);
                        lcd_data('.');
                        send_three_to_screen(dec_part);
                        lcd_data(' ');
                        lcd_data('\xdf');
                        lcd_data('C');
                }
        }
    }

}
```