Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

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

5ή εργαστηριακή άσκηση.

Ομάδα : 58
Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076
Ονοματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Για να είναι πιο οργανωμένος ο κώδικας μας, στην 5η Εργαστηριακή Άσκηση, χρησιμοποιήσαμε Header Files. Το header file και το Source File που υλοποιεί τις συναρτήσεις που ορίζουμε και στα 3 ζητήματα δίνονται παρακάτω:

```
#ifndef TWI_MACRO
#define TWI_MACRO

#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>


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_command (uint8_t input);
void write_2_nibbles_data (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
```

```c
#include "5.3.headers.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


//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_data (uint8_t input) {
	uint8_t temp = input & 0xF0;
	temp |= 0x04;

	temp |= 0x08;
	PCA9555_0_write(REG_OUTPUT_1, temp);
	asm("nop");
	asm("nop");
	temp &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_1, temp);


	temp = input & 0x0F;
```

```
	temp = temp << 4;
	temp |= 0x04;

	temp |= 0x08;
	PCA9555_0_write(REG_OUTPUT_1, temp);
	asm("nop");
	asm("nop");
	temp &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_1, temp);

	return;
}

void write_2_nibbles_command (uint8_t input) {
	uint8_t temp = input & 0xF0;
	//temp |= 0x04;

	temp |= 0x08;
	PCA9555_0_write(REG_OUTPUT_1, temp);
	asm("nop");
	asm("nop");
	temp &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_1, temp);


	temp = input & 0x0F;
	temp = temp << 4;
	//temp |= 0x04;

	temp |= 0x08;
	PCA9555_0_write(REG_OUTPUT_1, temp);
	asm("nop");
	asm("nop");
	temp &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_1, temp);

	return;
}


void lcd_data (uint8_t input) {
	write_2_nibbles_data(input);
	_delay_ms(1);
	return;
}
```

```
void lcd_command (uint8_t input) {
	write_2_nibbles_command(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 () {
	uint8_t dummy;
	_delay_ms(200);
	//PORTD = 0x30;
	PCA9555_0_write(REG_OUTPUT_1, 0x30);
	for (int i=0; i<3; i++) {
		dummy = 0x30;
		dummy |= 0x08;
		PCA9555_0_write(REG_OUTPUT_1, dummy);
		asm("nop");
		asm("nop");
		dummy &= ~(0x08);
		PCA9555_0_write(REG_OUTPUT_1, dummy);
		_delay_ms(1);
	}

	dummy = 0x20;
	dummy |= 0x08;
	PCA9555_0_write(REG_OUTPUT_1, dummy);
	asm("nop");
	asm("nop");
	dummy &= ~(0x08);
	PCA9555_0_write(REG_OUTPUT_1, dummy);
	_delay_ms(1);
```

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

}
```

Ζήτημα 5.1

Στο ζήτημα 5.1 μας ζητείται να υλοποιήσουμε 2 λογικές συναρτήσεις, με input από το PORTB και output στα LED_PD2 και LED_PD3, στα οποία όμως θα στέλνουμε την τιμή μέσω του PCA9555 και όχι μέσω του PORTD του μικροελεγκτή. Αφού θα χρησιμοποιήσουμε την θύρα 0 του PCA9555, χρησιμοποιώντας τις συναρτήσεις που μας παρέχονται από το εργαστήριο γράφουμε στον καταχωρητή REG_CONFIGURATION_0 την τιμή 0x00 (output), ενώ κάθε φορά που θα υπολογίζουμε τη συνάρτηση γράφουμε τα αποτελέσματα στον καταχωρητή REG_OUTPUT_0. Τέλος, για να παρατηρούμε το αποτέλεσμα, ενώνουμε με καλώδια τους ακροδέκτες IO0_0 και IO0_1 του κονέκτορα P18 με τους ακροδέκτες LED_PD2 και LED_PD3 του κονέκτορα J18 αντίστοιχα. Παρακάτω ακολουθέι το main file που υλοποιεί το παραπάνω ζήτημα.

```
#include "51headers.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

int main() {
	//Set PORTB[3:0] as input
	DDRB = 0x00;
	//Set PORTD as output in order to connect PD2 and PD3 with IO0_0
and IO0_1
	DDRD = 0xFF;
	//DDRC = 0xFF;
	unsigned char A,B,C,D, F0, F1, input, result;
	twi_init();
	PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as
output
```

```
	while(1) {
		input = PINB;

		A = input & 0x01;
		B = (input & 0x02) >> 1;
		C = (input & 0x04) >> 2;
		D = (input & 0x08) >> 3;
		F0 = ~( ((~A) & B & C) | (~(B) & D));
		F0 &= 0x01;
		F1 = (A | B | C) & (B & (~D));
		F1 = (F1 & 0x01) << 1;
		result = F0 | F1;
		PCA9555_0_write(REG_OUTPUT_0, result);

	}
}
```

Ζήτημα 5.2

Στο 5.2 μας ζητείται να γράψουμε κώδικα ο οποίος θα διαβάζει input από το πληκτρολόγιο και ανάλογα με το ποιο πλήκτρο της 1ης γραμμής του έχει πατηθεί θα ανάβει (χρησιμοποιώντας πάλι το PCA9555 και όχι το PORTD) το κατάλληλο LED_PD. Ορίζουμε λοιπόν ως έξοδο τον ακροδέκτη IO1_0 της θύρας επέκτασης 1 του PCA9555, ενώ ορίζουμε ως είσοδο τους ακροδέκτες IO1_4 με IO1_7. Για να διαβάσουμε από την πρώτη γραμμή του πληκτρολογίου, γράφουμε λογικό 0 στο LSB του REG_OUTPUT_1, και μετά διαβάζουμε συνεχόμενα το high nibble του καταχωρητή REG_INPUT_1, όταν ένα bit είναι 0 σημαίνει ότι είναι πατημένο το αντίστοιχο πλήκτρο (αφού κλείνει ο διακόπτης και η λογική τιμή του IO1_0 περνάει πάνω στον αντίστοιχο ακροδέκτη). Τέλος, όπως και στο πρώτο ζήτημα, φροντίζουμε να συνδέσουμε με καλώδια τους κατάλληλους ακροδέκτες IO0 με τους κατάλληλους ακροδέκτες LED_PD, ώστε να μπορούμε μέσω του ολοκληρωμένου PCA9555 να ανάβουμε τα LEDs που μας ζητείται. Παρακάτω ακολουθεί ο κώδικας του main file:

```
#include "5.3.headers.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

int main() {
    twi_init();
```

```
    //Set Port Expander 0 as output
    PCA9555_0_write(REG_CONFIGURATION_0,0x00);
    //Set Port Expander 1 [3:0] as output ((and [7:4] as input))
    PCA9555_0_write(REG_CONFIGURATION_1,0xFE);
    uint8_t input;
    //DDRD = 0xFF;
    PCA9555_0_write(REG_OUTPUT_1,0x00); //maybe


    while(1)
    {
        //PCA9555_0_write(REG_OUTPUT_0, 0xFF);
        input = PCA9555_0_read(REG_INPUT_1);


        if((input & 0x10) == 0x00) //*
            // WE ALWAYS GO HERE
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x01);
        }
        else if((input & 0x20) == 0x00) // 0
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x02);
        }
        else if((input & 0x40) == 0x00) // #
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x04);
        }
        else if((input & 0x80) == 0x00) // D
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x08);
        }
        else
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x00);
        }
    }
    return 0;
}
```

Ζήτημα 5.3

Στο 5.3 μας ζητείται να τυπώνουμε τα ονόματα μας στην LCD οθόνη, αλλά αντίθετα με την 4η άσκηση να χρησιμοποιούμε την θύρα επέκτασης 1 του ολοκληρωμένου PCA9555 αντί για το PORTD, για την επικοινωνία με την οθόνη. Θα προσθέσουμε λοιπόν στο header file μας, τις συναρτήσεις που μας είχαν δοθεί στην 4η εργαστηριακή, κατάλληλα τροποποιημένες. Ακόμα θα γράψουμε και μία επιπλέον συνάρτηση lcd_change_line, η οποία θα κάνει shift την DDRAM στην θέση 0x40, που γνωρίζουμε ότι βρίσκεται η δεύτερη γραμμή. Τέλος, θα φροντίσουμε να ενώσουμε τους ακροδέκτες της οθόνης με αυτούς του ολοκληρωμένου, στον κονέκτορα J19.Παρακάτω φαίνεται το main file μας :

```
#define F_CPU 16000000UL
#include "5.3.headers.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


int main() {
	twi_init();
	PCA9555_0_write(REG_CONFIGURATION_1, 0x00); //Set EXT_PORT0 as
output
	lcd_init();

	char fezz1[] = "Konstantinos", fezz2[] = "Fezos", oar1[] =
"Odysseas", oar2[] = "Tsouknidas";
	while (1) {
		lcd_clear_display();
		for (int i=0; i<12; i++){
			lcd_data(fezz1[i]);
		}
		lcd_change_line();
		for (int i=0; i<5; i++){
			lcd_data(fezz2[i]);
		}


		_delay_ms(50000);
		// Now time for the next name
		lcd_clear_display();
		for (int i=0; i<8; i++) {
			lcd_data(oar1[i]);
```

```
            }
            lcd_change_line();
            for (int i=0; i<10; i++){
                  lcd_data(oar2[i]);
            }
            _delay_ms(50000);

      }

      return 0;
}
```