Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

5η Σειρά Ασκήσεων

Μάθημα: Εργαστήριο Μικροϋπολογιστών

Εξάμηνο: 7ο

Ονοματεπώνυμο: Αλεξοπούλου Γεωργία), Γκενάκου Ζωή

Ζήτημα 5.1

Να υλοποιηθούν για το μικροελεγκτή ATmega328PB, σε γλώσσα C, οι λογικές συναρτήσεις:
F0 = (A'B + B'CD)'
F1= (AC)(B+D)
Οι μεταβλητές εισόδου δίνονται στα bit **PORTB [3:0]** (A= PORTB.0, B= PORTB.1, C= PORTB.2, D= PORTB.3) του ntuAboard_G1. Οι τιμές των F0- F1 να εμφανίζονται αντίστοιχα στους ακροδέκτες IO0_0 και IO0_1 του ολοκληρωμένου επέκτασης θυρών PCA9555, στο ntuAboard_G1. Οι ακροδέκτες IO0_0 και IO0_1 του κονέκτορα P18 να συνδεθούν, μέσω καλωδίων, με τους ακροδέκτες LED_PD0 και LED_PD1 του κονέκτορα J18 αντίστοιχα, έτσι ώστε οι τιμές των συναρτήσεων F0 και F1 να εμφανίζονται στα led PD0 PD1.

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

```c
#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;
}
// 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));
}
unsigned char twi_readNak(void)
{
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));

    return TWDR0;
}
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;
}

uint8_t A, B, C, D, A_inverse, B_inverse, F0, F1, F0_inverse, output;

uint8_t inverse(uint8_t x)
{
    if (x == 0) x = 1;
    else x = 0;

    return x;
}
```

```c
int main(void)
{
    twi_init();
    DDRB = 0x00;                                 // Initialize PORTB as
input
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);      // Set EXT_PORT0 as
output

    while (1)
    {

        uint8_t input = PINB;

        A = input & 0x01;
        B = (input  & 0x02) >> 1;
        C = (input  & 0x04) >> 2;
        D = (input  & 0x08) >> 3;
        A_inverse = inverse(A);             //A'
        B_inverse = inverse(B);             //B'

        //  F0 = (A'B + B'CD)'
        F0= ((A_inverse && B) | (B_inverse && (C && D)));
        F0_inverse = inverse(F0);

        //  F1 = (AC)(B+D)
        F1 = ((A && C) && (B || D));

        F1 = F1 << 1;
        output = F0_inverse + F1 ;
        PCA9555_0_write(REG_OUTPUT_0, output );

    }
}


//A     B       C       D       F0      F1
//0     0       0       0       1       0
//0     0       0       1       1       0
//0     0       1       0       1       0
//0     0       1       1       0       0
//0     1       0       0       0       0
//0     1       0       1       0       0
//0     1       1       0       0       0
```

```
//0     1       1       1       0       0
//1     0       0       0       1       0
//1     0       0       1       1       0
//1     0       1       0       1       0
//1     0       1       1       0       1
//1     1       0       0       1       0
//1     1       0       1       1       0
//1     1       1       0       1       1
//1     1       1       1       1       1
```

Ο παραπάνω κώδικας έχει διαμορφωθεί με τη χρήση του δοθέντος παραδείγματος. Στην κύρια συνάρτηση αριχοποιούμε τα ζητούμενα inputs και αντιστρέφουμε τα A και B, με τη χρήση της συνάρτησης `uint8_t inverse`. Στη συνέχεια, χρησιμοποιώντας τις λογικές πράξεις && (AND gate) και || (OR gate), σχηματίζουμε τις συναρτήσεις F0, F1 που περιγράφονται στην εκφώνηση. Με την εντολή `PCA9555_0_write(REG_OUTPUT_0, output );` εμφανίζουμε τις τιμές των F0, F1 στο PORTD (PD0, PD1 αντίστοιχα). Όταν το εκάστοτε λαμπάκι είναι αναμμένο, τότε η τιμή της συνάρτησης ισούται με 1, ενώ όταν είναι σβηστό ισούται με 0. Όσον αφορά στο PORTB, όταν δεν πιέζουμε κάποιο από τα PB0-PB3, η τιμή της αντίστοιχης εισόδου ισούται με 1 και το λαμπάκι είναι αναμμένο. Αντίθετα, όταν πιέζουμε κάποιο από αυτά τα κουμπιά, η τιμή της αντίστοιχης εισόδου ισούται με 0 και το λαμπάκι σβήνει.

Ζήτημα 5.2

Προκειμένου να μπορέσει να υπάρξει οπτική απεικόνιση της λογικής κατάστασης των ακροδεκτών IO0_0 έως IO0_3, της θύρας επέκτασης 0, να ρυθμιστούν ως έξοδοι και να συνδεθούν, μέσω καλωδίων, με τους ακροδέκτες LED_PD0 έως LED_PD3 του κονέκτορα J18 αντίστοιχα.
Ο ακροδέκτης IO1_0 της θύρας επέκτασης 1 να ρυθμιστεί ως έξοδος ενώ οι ακροδέκτες IO1_4 έως IO0_7 να ρυθμιστούν ως είσοδοι.
Να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος όταν πιέζεται το πλήκτρο “*” να ανάβει το led PD0, όταν πιέζεται το πλήκτρο “0” να ανάβει το led PD 1, όταν πιέζεται το πλήκτρο “#” να ανάβει το led PD2 και όταν πιέζεται το πλήκτρο “D” να ανάβει το led PD3. Αν δεν πιέζεται κανένα πλήκτρο τότε όλα τα led να παραμένουν σβηστά.

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

#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
```

```c
#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
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;
}


// 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;
    }
```

```c
}

// 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));
}

unsigned char twi_readNak(void)
{
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));

    return TWDR0;
}

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;
}

int main(void){

    twi_init();
    DDRB = 0x00;                                // Initialize PORTB as
input

    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);     // Set EXT_PORT0 as
output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xFE);     // Set IO1_0 as output
and others as input

    while(1)
    {
        PCA9555_0_write(REG_OUTPUT_1, 0xFE);
        PCA9555_0_write(REG_OUTPUT_0, 0x00);        // Turn off LEDs

        uint8_t input = PCA9555_0_read(REG_INPUT_1);

        if (input == 0xEE)
            PCA9555_0_write(REG_OUTPUT_0, 0x01);

        if (input == 0xDE)
            PCA9555_0_write(REG_OUTPUT_0, 0x02);
```

```
        if (input == 0xBE)
            PCA9555_0_write(REG_OUTPUT_0, 0x04);

        if (input == 0x7E)
            PCA9555_0_write(REG_OUTPUT_0, 0x08);
    }
}
```

Ακολουθώντας, και πάλι, τον σκελετό του δοθέντος παραδείγματος, στην κύρια συνάρτηση του παραπάνω κώδικα εξετάζουμε τις 4 διαφορετικές περιπτώσεις. Όταν η είσοδος από το πληκτρολόγιο αντιστοιχεί σε έναν από τους κωδικούς των πλήκτρων *, 0, #, D, τότε ανάβει το λαμπάκι του PD0, PD1, PD2, PD3 αντίστοιχα. Αξίζει να σημειωθεί πως, όταν 2 ή περισσότερα εξ' αυτών των πλήκτρων πιέζονται ταυτόχρονα, τότε δεν ανάβει κανένα από τα PIND.

Ζήτημα 5.3

Να συνδεθεί η LCD οθόνη 2x16 χαρακτήρων, δια μέσω του κονέκτορα J19, στην θύρα επέκτασης 1 του ολοκληρωμένου PCA9555 και να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος θα απεικονίζει στην οθόνη το όνομα και το επίθετο σας

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

uint8_t buffer;

//initialize TWI clock
void twi_init(void)
{
    TWSR0 = 0; // PRESCALER_VALUE=1
    TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}
```

```c
// 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;
}


// 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));
}

unsigned char twi_readNak(void)
{
```

```
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));

    return TWDR0;
}

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){

    uint8_t temp = input;
    uint8_t pin = buffer;
    pin &= 0x0F;
    input &=0xF0;
    input |= pin;
    buffer = input;
    buffer |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    buffer &= 0xF7;
```

```
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    input = temp;
    input &= 0x0F;
    input = input << 4;
    input |= pin;
    buffer = input;
    buffer |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    buffer &= 0xF7;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    return;
}

void LCD_data(uint8_t x){

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

    write_2_nibbles(x);
    _delay_us(250);
    return;

}

void LCD_command(uint8_t x){

    buffer &= 0xFB;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    write_2_nibbles(x);
    _delay_us(250);
    return;

}

void LCD_clear_display(){
```

```c
    uint8_t x = 0x01;
    LCD_command(x);
    _delay_ms(5);

}


void LCD_init(void){

    _delay_ms(200);

    buffer = 0x30;
    buffer |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    buffer &= 0xF7;
    PCA9555_0_write(REG_OUTPUT_0, buffer);
    _delay_us(250);

    buffer = 0x30;
    buffer |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    buffer &= 0xF7;
    PCA9555_0_write(REG_OUTPUT_0, buffer);
    _delay_us(250);

    buffer = 0x20;
    buffer |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0, buffer);

    buffer &= 0xF7;
    PCA9555_0_write(REG_OUTPUT_0, buffer);
    _delay_us(250);

    LCD_command(0x28);
    LCD_command(0x0C);
```

```c
    LCD_clear_display();

    LCD_command(0x06);

}

int main(void)
{
    twi_init();

    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);     // Set EXT_PORT0
as output

    LCD_init();

    while(1){
        LCD_data('Z');
        LCD_data('O');
        LCD_data('E');
        LCD_data(' ');
        LCD_data('G');
        LCD_data('K');
        LCD_data('E');
        LCD_data('N');
        LCD_data('A');
        LCD_data('K');
        LCD_data('O');
        LCD_data('U');

        _delay_ms(2000);
        LCD_clear_display();

        LCD_data('G');
        LCD_data('I');
        LCD_data('N');
        LCD_data('A');

        LCD_data(' ');
```

```
        LCD_data('A');
        LCD_data('L');
        LCD_data('E');
        LCD_data('X');
        LCD_data('O');
        LCD_data('P');
        LCD_data('O');
        LCD_data('U');
        LCD_data('L');
        LCD_data('O');
        LCD_data('U');

        _delay_ms(2000);
        LCD_clear_display();

    }

}
```

Συνδυάζοντας τους κώδικες των παραδειγμάτων 4 και 5, λαμβάνουμε τον παραπάνω κώδικα. Δημιουργούμε μια global μεταβλητή ονόματι buffer, στην οποία αποθηκεύονται όλες οι αλλαγές που γίνονται στο output του PCA9555. Η πραγματική αλλαγή σε σχέση με το παράδειγμα 4, όπου η οθόνη δεν είναι συνδεδεμένη με το ολοκληρωμένο PCA9555, είναι η αντικατάσταση του PORTD με τον buffer, και μετά από οποιαδήποτε αλλαγή στον buffer αξιοποιούμε την εντολή `PCA9555_0_write(REG_OUTPUT_0, buffer)`. Στην κύρια συνάρτηση, αρχικοποιούμε τόσο τον PCA9555, όσο και την LCD οθόνη κι εκτυπώνουμε τα ονόματά μας, έναν χαρακτήρα την φορά με την συνάρτηση `LCD_data`.