



Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ  
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ  
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ  
ΑΚΑΔ. ΕΤΟΣ 2022-2023

ΑΘΗΝΑ, 23 Νοεμβρίου 2022

**5<sup>η</sup> ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**  
**ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"**  
**Χρήση εξωτερικών Θυρών Επέκτασης στον AVR**

**Αναφορά 5<sup>ης</sup> Εργαστηριακής Άσκησης**

**Ραπτόπουλος Πέτρος (el19145)**  
**Σαφός Κωνσταντίνος (el19172)**

## Ζήτημα 5.1:

Παρακάτω φαίνεται ο κώδικας σε C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

**Σημείωση:** Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

### Κώδικας σε C

```
#define F_CPU 16000000UL
#include <math.h>
#include <util/delay.h>
#include <avr/io.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
//Fsc1=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) {
    TWCRR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while(!(TWCRR0 & (1<<TWINT)));
    return TWDRO;
}

// 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;
    TWCRR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN); // send START condition
    while(!(TWCRR0 & (1<<TWINT))); // wait until transmission completed
    twi_status = TW_STATUS & 0xF8; // check value of TWI Status Register.
    if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
    TWDRO = address; // send device address
    TWCRR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCRR0 & (1<<TWINT))); // wait until transmission completed and ACK/NACK has been received
    twi_status = TW_STATUS & 0xF8; // check value of TWI Status Register.
    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) {
        TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);    // send START condition
        while(!(TWCR0 & (1<<TWINT)));                    // wait until transmission completed
        twi_status = TW_STATUS & 0xF8;                  // check value of TWI Status Register.
        if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;
        TWDRO = address;                                // send device address
        TWCR0 = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR0 & (1<<TWINT)));                    // wait until transmission completed
        twi_status = TW_STATUS & 0xF8;                  // check value of TWI Status Register.
        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);
            while(TWCR0 & (1<<TWSTO)); // wait until stop condition is executed and bus released
            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) {
    TWDRO = data;    // send data to the previously addressed device
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT))); // wait until transmission completed
    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) {
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);    // send stop condition
    while(TWCR0 & (1<<TWSTO)); // wait until stop condition is executed and bus released
}

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 twi_readNak(void) {
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDRO;
}

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() {
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    DDRB = 0x00; //PORTB input
    char A, B, C, D;
    while(1) {
        char state = ~PINB;
        A = state & 0x01;
        B = state & 0x02;
        B = B >> 1;
        C = state & 0x04;
        C = C >> 2;
        D = state & 0x08;
        D = D >> 3;

        char F = ~(((~A) & B) | ((~B) & C & D));
        char G = ((A & C) & (B | D));
        G = (G << 1)&0x02;
        F = F&0x01;
        PCA9555_0_write(REG_OUTPUT_0, (F | G));
    }
}

```

## Ζήτηση 5.2:

Παρακάτω φαίνεται ο κώδικας σε C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

**Σημείωση:** Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

```

int main() {
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1_0 as output, 4-7 as input
    PCA9555_0_write(REG_OUTPUT_1, 0x0E);
    char button, leds;
    while(1) {
        leds = 0;
        button = PCA9555_0_read(REG_INPUT_1);
        if ((button & 0x10) == 0x00) {
            leds |= 0x01;
        }
        if ((button & 0x20) == 0x00) {
            leds |= 0x02;
        }
        if ((button & 0x40) == 0x00) {
            leds |= 0x04;
        }
        if ((button & 0x80) == 0x00) {
            leds |= 0x08;
        }
        PCA9555_0_write(REG_OUTPUT_0, leds);
    }
}

```

**Κώδικας σε C**

Σημείωση: Οι υπόλοιπες βοηθητικές συναρτήσεις παραμένουν οι ίδιες με αυτές της προηγούμενης άσκησης. Συνεπώς δεν ξαναπαρουσιάζονται στην αναφορά.