ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (MICROLAB)

5η Εργαστηριακή Αναφορά στο μάθημα “**ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ”** του 7ου Εξαμήνου

των φοιτητών της **ομάδας 17**,

Εμμανουήλ Αναστάσιου Σερλή, Α.Μ. 03118125
Ευάγγελου Περσόπουλου, Α.Μ.: 03121701

1η Άσκηση: -> micro_lab05_ex01.c

Το ζητούμενο πρόγραμμα υλοποιεί τις λογικές συναρτήσεις F0 και F1 της εκφώνησης, με τις μεταβλητές εισόδου να δίνονται από τα pins εισόδου PB0 ως και PB3. Οι έξοδοι των λογικών συναρτήσεων εμφανίζονται στα pins εξόδου IO00 και IO01, και --στην συνέχεια μέσω jumper wires- στα PINS 0 και 1 του PORTD αντίστοιχα. Οι αναμενόμενες τιμές των λογικών συναρτήσεων για τους διαφορετικούς συνδυασμούς των μεταβλητών A,B,C,D αναγράφονται στον κάτωθι πίνακα:

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	0
1	1	0	0	1	0
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

C Program:

```
#define F_CPU 16000000UL //running
#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)
//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;
}

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));
}
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) {
 DDRB |=0b00000000;
 twi_init();

 PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
 while(1){
 asm("NOP");
 char temp0= ((~PINB)&(0x01));
 char temp1=((~PINB)&(0x02))>>1; //get PORTB lsb values
 char temp2=((~PINB)&(0x04))>>2;
 char temp3=((~PINB)&(0x08))>>3;

 char temp5=!( ((!temp0)&&temp1) || ((!temp1)&&temp2&&temp3 ) );
//logic functions
 char temp6 = ((temp0&&temp2)&&(temp1||temp3));

 if (temp6==0x00 && temp5==0x00){ //different led combos according to
functions' output
     asm("NOP");
      PCA9555_0_write(REG_OUTPUT_0, 0x00);
 }
 if (temp6==0x00 && temp5==0x01){
     asm("NOP");
      PCA9555_0_write(REG_OUTPUT_0, 0x01);
 }
 if (temp6==0x01 && temp5==0x00){
     asm("NOP");
      PCA9555_0_write(REG_OUTPUT_0, 0x02);
 }
  if (temp6==0x01 && temp5==0x01){
     asm("NOP");
      PCA9555_0_write(REG_OUTPUT_0, 0x03);
 }
}
 return 0;
}
```

2η Άσκηση: -> micro_lab04_ex02.c

Το ζητούμενο πρόγραμμα ανάβει το κατάλληλο LED (PD0-PD3) στο PORTD ανάλογα με το πλήκτρο του keyboard το οποίο πιέζεται, ενώ σε περίπτωση που δεν πιέζεται κάποιο πλήκτρο τότε δεν ανάβει κάποιο από τα leds. Ο ακροδέκτης

IO1_0 του ολοκληρωμένου PCA9555 έχει ρυθμιστεί ως έξοδος ενώ οι ακροδέκτες IO1_4 ως IO1_7 ως είσοδοι.

<u>C Program:</u>

```
#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)
//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;
}

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));
}
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)
{
    uint8_t keyboard;
    PORTD |= 0b11111111;

    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1's
bit4-7 as input and bit0 as output
    PCA9555_0_write(REG_INPUT_1, 0xFE); //write 0 for row 1
    while(1)
    {
        keyboard = PCA9555_0_read(REG_INPUT_1); //read IO1 pins
        _delay_ms(10);

        if (keyboard == 0b11101110) //if pressed "*"
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x01);    //flash LED0
            _delay_ms(10);
        }
        if (keyboard == 0b11011110) //if pressed "0"
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x02);    //flash LED1
            _delay_ms(10);
        }
        if (keyboard == 0b10111110) //if pressed "#"
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x04);    //flash LED2
            _delay_ms(10);
        }
        if (keyboard == 0b01111110) //if pressed "D"
        {
            PCA9555_0_write(REG_OUTPUT_0, 0x08);    //flash LED3
            _delay_ms(10);
        }
        PCA9555_0_write(REG_OUTPUT_0, 0x00);
    }
    return 0;
}
```