**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών**
**Εργαστήριο Μικροϋπολογιστών**

Έκθεση 5ης Εργαστηριακής Άσκησης

Στοιχεία:
Ομάδα: **39**
Μέλη: **Ευάγγελος Μυργιώτης, Αριστοτέλης Γρίβας**

Ζήτημα 5.1 :

Το ζητούμενο της άσκησης είναι 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 πρέπει να εμφανίζονται αντίστοιχα στους ακροδέκτες IO0_0 και IO0_1.

Συνεπώς,στον παρακάτω κώδικα,παίρνουμε την τιμή της εισόδου στη μεταβλητή x με την εντολή “x =~PINB”(αντίστροφη λογική),και στη συνέχεια αποθηκεύουμε στα LSBs των μεταβλητών A,B,C,D την κατάσταση των τεσσάρων πρώτων buttons που έχουμε πατημένα αντίστοιχα.
Έπειτα μέσω των bitwise operations της C υλοποιούμε τις λογικές συναρτήσεις και αποθηκεύουμε τα αποτελέσματά τους στα LSBs των μεταβλητών F0 και F1 αντίστοιχα.
Τέλος,κάνουμε shift left μία φορά το F1 ώστε να πάει το αποτέλεσμά του στη θέση του δεύτερου LSB και το προσθέτουμε στη μεταβλητή F1 που έχει στο LSB του το αποτέλεσμα F0 και τυπώνουμε το F0 στην PORTD.
Η τελική μορφή του F0 είναι όπως περιγράψαμε(0b000000yz),όπου
y = F1 , z = F0.

Κώδικας σε C :

```
#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

unsigned char A,B,C,D,temp1,temp2,x,F1,F0;




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) {
 twi_init();

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

 while(1)
 {

 x = ~PINB;

 A = x & 1;
 B = (x >> 1) & 1;
 C = (x>>2) & 1;
 D = (x>>3) & 1;

 temp1 = A;
 temp2 = B;

 F0 = ( (A | (~B)) & (B | (~C) | (~D)));
 F1 = ( (A & C) & (B | D) ) << 1;
 F0 = F0 & 1;
 F1 = F1 & 2;
 F0 = F0 | F1;

 PCA9555_0_write(REG_OUTPUT_0, F0);

 }
}
```

Ζήτημα 5.2 :

Το ζητούμενο της άσκησης είναι 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.
Συνεπώς,στον παρακάτω κώδικα θέτουμε τα IO1.0 εως IO1.3 ως εξόδους και δίνουμε τις τιμες 0 στο IO1.0 και 1 στα υπόλοιπα, ώστε να διαβάσουμε μονό την πρώτη γραμμή θέτοντας τα IO1.4 έως IO1.7 ως είσοδοι.
Στη συνέχεια διαβάζουμε την είσοδο μέσω της συνάρτησης read() και ελέγχοντας κάθε φορά το πλήκτρο που πατάμε,ανάβουμε το αντίστοιχο LED στην έξοδο της PORTD.Οι εντολές while() χρησιμεύουν ώστε να μην μπορούμε να πατήσουμε παραπάνω από ένα πλήκτρα ταυτόχρονα.Αν δεν πατάμε κανένα κουμπί η εντολή else θα σβήσει τα LEDs.

Κώδικας σε C :

```
#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

unsigned char A,B,C,D,temp1,temp2,x,F1,F0;
uint8_t y;




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) {

 twi_init();
```

```
PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
PCA9555_0_write(REG_OUTPUT_1, 0xFE);


while(1)
{
y =  ~PCA9555_0_read(REG_INPUT_1);
if(y & 0x10){
    PCA9555_0_write(REG_OUTPUT_0, 0x01);
    while(1){
        if (PCA9555_0_read(REG_INPUT_1) & 0x10)
            break;

    }
}


else if(y & 0x20){
    PCA9555_0_write(REG_OUTPUT_0, 0x02);
    while(1){
        if (PCA9555_0_read(REG_INPUT_1) & 0x20)
            break;

    }
}

else if(y & 0x40){
    PCA9555_0_write(REG_OUTPUT_0, 0x04);
    while(1){
        if (PCA9555_0_read(REG_INPUT_1) & 0x40)
            break;

    }
}

else if(y & 0x80){
    PCA9555_0_write(REG_OUTPUT_0, 0x08);
    while(1){
        if (PCA9555_0_read(REG_INPUT_1) & 0x80)
            break;

    }
}

else{
    PCA9555_0_write(REG_OUTPUT_0, 0x00);
}

}
}
```

Ζήτημα 5.3 :

Το ζητούμενο της άσκησης είναι να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C, ο οποίος θα απεικονίζει στην οθόνη το όνομα και το επίθετο.Ωστόσο,επειδή η ομάδα μας αποτελείται από 2 άτομα,τροποποιήσαμε τον κώδικα ώστε να :

.Τυπώνει συνεχόμενα στην LCD οθόνη των αριθμό της ομάδας μας(ομάδα 39)
.Με συνεχόμενο  πάτημα του κουμπιού PC0 να τυπώνει το όνομα του ενός μέλους στην πρώτη γραμμή της οθόνης και το επίθετό του στη δεύτερη γραμμή
. Με συνεχόμενο  πάτημα του κουμπιού PC1 να τυπώνει το όνομα του άλλου μέλους στην πρώτη γραμμή της οθόνης και το επίθετό του στη δεύτερη γραμμή

Παραθέτουμε τον κώδικα παρακάτω:

Κώδικας  C :

```
#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

unsigned char A,B,C,D,temp1,temp2,x,F1,F0;




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;
}
void write_2_nibbles(unsigned char q){
```

```
    int temp = q;   // r24 = temp = q
    int e = PCA9555_0_read(REG_OUTPUT_0);   //in r25, PIND
    e = e & 0x0f;    // andi r25, 0x0f
    temp = temp & 0xf0; // andi r24, 0xf0
    temp = temp + e;    // add r24, r25
      //out PORTD ,r24
    PCA9555_0_write(REG_OUTPUT_0,temp);  //out PORTD ,r24
    temp |= 0x08;  //sbi PORTD ,3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);
    temp &= 0b11110111; //cbi PORTD ,3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);

    temp = q;   //pop r24
    temp = temp<<4;  //swap r24
    temp = temp & 0xf0;  //andi r24, 0xf0
    temp = temp + e;   //add r24,r25
    PCA9555_0_write(REG_OUTPUT_0,temp);   //portd, r24
    temp |= 0x08;
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);
    temp &= 0b11110111;
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(2);

return;
}


void lcd_data(unsigned char p){
    int temp = PCA9555_0_read(REG_OUTPUT_0);
    temp |= 0x04;
    PCA9555_0_write(REG_OUTPUT_0,temp);
    write_2_nibbles(p);
    _delay_ms(1);
    return;
}

void lcd_command(unsigned char z){
    unsigned char temp = PCA9555_0_read(REG_OUTPUT_0);
    temp &= 0b11111011;
    PCA9555_0_write(REG_OUTPUT_0,temp);
    write_2_nibbles(z);
    _delay_ms(1);
    return;
}


void lcd_init(){
    unsigned char temp;
    _delay_ms(40); //wait_msec

        PCA9555_0_write(REG_OUTPUT_0,0x30);   //PORTD = r24 = 0x30
```

```
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp |= 0x08;   // sbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp &= 0b11110111; //cbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(2);

     PCA9555_0_write(REG_OUTPUT_0,0x30);   //PORTD = r24 = 0x30
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp |= 0x08;   // sbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp &= 0b11110111; //cbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(2);

    PCA9555_0_write(REG_OUTPUT_0,0x30);   //PORTD = r24 = 0x30
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp |= 0x08;   // sbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp &= 0b11110111; //cbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(2);

   PCA9555_0_write(REG_OUTPUT_0,0x20);   //PORTD = r24 = 0x30
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp |= 0x08;   // sbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(1);
    temp = PCA9555_0_read(REG_OUTPUT_0);
    temp &= 0b11110111; //cbi PORTD, 3
    PCA9555_0_write(REG_OUTPUT_0,temp);
    _delay_ms(2);


    lcd_command(0x28);
    lcd_command(0x0c);
    lcd_command(0x01);

    _delay_ms(10);

    lcd_command(0x06);

    _delay_ms(10);

    return;
}
```

```
int main(void) {
 twi_init();
 uint8_t onoma[11] = "Aristotelis";
 uint8_t eponimo[6] = "Grivas";
 uint8_t onoma2[9] = "Evangelos";
 uint8_t eponimo2[9] = "Mirgiotis";
 uint8_t omada[6] = "Omada:";
 uint8_t noumero[6] = "No. 39";
 unsigned char o,e;

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

 while(1)
 {
    lcd_init();

    for(int i = 0; i <11; i++){

        lcd_data(onoma[i]);

    }

    lcd_command(0b11000000);

    for(int i = 0; i <6; i++){
        lcd_data(eponimo[i]);
    }


   _delay_ms(1000);

    lcd_init();

   for(int i = 0; i <9; i++){

        lcd_data(onoma2[i]);

    }

    lcd_command(0b11000000);

    for(int i = 0; i <9; i++){
        lcd_data(eponimo2[i]);
    }


   _delay_ms(1000);

   lcd_init();

   for(int i = 0; i <6; i++){
```

```
        lcd_data(omada[i]);

    }

    lcd_command(0b11000000);

    for(int i = 0; i <6; i++){
        lcd_data(noumero[i]);
    }


   _delay_ms(1000);
 }
}
```