Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

Έκθεση 6ης Εργαστηριακής Άσκησης

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

Ζήτημα 6.1 :

Η ιδέα της υλοποίησης των συναρτήσεων είναι η εξής:

-void scan_row(int r, int a[], uint8_t c) :

Η συνάρτηση αυτή έχει 3 ορίσματα όπου r η γραμμή που σκανάρουμε, c ο ακροδέκτης που ρυθμίζουμε κάθε φορά ως έξοδο για να σκανάρουμε την αντίστοιχη γραμμή του πληκτρολογίου και α[] ένας πίνακας στον οποίο θα ανανεώνουμε τις καταστάσεις των πλήκτρων(βάζουμε τιμές 1 και 0,όπου για 1 έχουμε πατημένο πλήκτρο και 0 μη πατημένο).

-void scan_keypad(int a[]):

Η συνάρτηση αυτή καλεί τη συνάρτηση scan_row 4 φορές,σκανάροντας και τις 4 γραμμές του πληκρολογίου και αποθηκεύει τις καταστάσεις των πλήκτρων στον πίνακα a[] όπως εξηγήσαμε παραπάνω.

-void scan_keypad_rising_edge() :

Η συνάρτηση αυτή καλεί τη scan_keypad μία φορά και αποθηκεύει τις καταστάσεις των πλήκτρων στον πίνακα key[],μετά από 20 ms(καθυστέρηση για να λάβουμε υπόψη το σπινθηρισμό),καλεί και πάλι τη scan_keypad και αποθηκεύει τις νέες καταστάσεις των πλήκτρων στον πίνακα pad[].
Στη συνέχεια συγκρίνει τις 2 καταστάσεις του κάθε πλήκτρου από τους πίνακες key[] και pad[] και:
.Αν έχουμε διαφορετική κατάσταση για κάποια τιμή στους δύο πίνακες αυτό σημαίνει αλλαγή στην κατάσταση του πλήκτρου και συνεπώς πάτημα κουμπιού. Άρα, αποθηκεύουμε στον πίνακα rise[] την τιμή 1 για αυτό το κουμπί.
.Διαφορετικά αν οι πίνακες έχουν την ίδια τιμή,αυτό σημαίνει ότι η κατάσταση παρέμεινε ίδια,άρα δεν πατήσαμε το αντίστοιχο κουμπί. Άρα, αποθηκεύουμε στον πίνακα rise[] την τιμή 0 για αυτό το κουμπί.

-uint8_t keypad_to_ascii(int a[]):

Τέλος αυτός ο πίνακας παίρνει ως όρισμα έναν πίνακα ο οποίος έχει την πληροφορία των κουμπιών που είναι πατημένα και αναλόγως το κουμπί που πατήσαμε,δίνει στην έξοδο τον κωδικό ascii του μέσω του πίνακα keypad[].

Στη main(),αρχίζουμε τυπώνοντας τον αριθμό 0 στην lcd οθόνη,και με το στη συνέχεια σκανάρουμε για πάτημα κουμπιού με τη συνάρτηση scan_keypad_rising_edge.Αν κάποιο κουμπί πατήθηκε,τότε παίρνουμε τον κωδικό ascii του μέσω της keypad_to_ascii και τυπώνουμε το αποτέλεσμα στην lcd οθόνη έως ότου πατήσουμε κάποιο άλλο κουμπί.

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;
unsigned char keypad[16] = { 0b00101010, 0b00110000, 0b00100011, 0b01000100, 0b00110111,
0b00111000, 0b00111001, 0b01000011, 0b00110100, 0b00110101, 0b00110110, 0b01000010,
0b00110001, 0b00110010, 0b00110011, 0b01000001};



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;
}
```

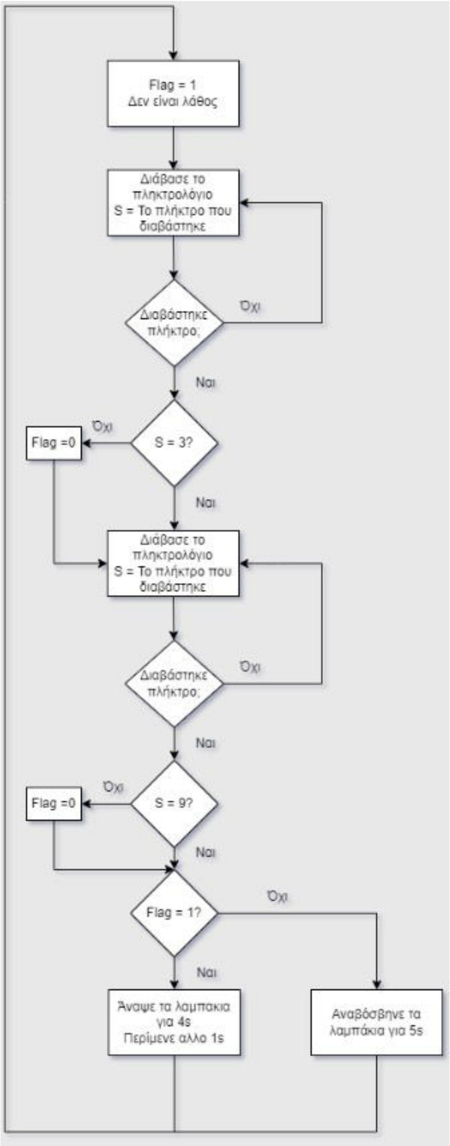
```
// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address)
{
 uint8_t twi_status;
 while ( 1 )
 {
// send START condition
TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

 // 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 key[16];
int pad[16];
int rise[16];

void  scan_row(int r, int a[], uint8_t c){
uint8_t y = 0x10;

        PCA9555_0_write(REG_OUTPUT_1, c);

    y =  ~PCA9555_0_read(REG_INPUT_1);

    a[r*4 + 0] = y & 0x10;
    a[r*4 + 1] = y & 0x20;
    a[r*4 + 2] = y & 0x40;
    a[r*4 + 3] = y & 0x80;

}



void scan_keypad(int a[]){
    scan_row(0,a,0xFE);
    scan_row(1,a,0xFD);
    scan_row(2,a,0xFB);
    scan_row(3,a,0xF7);
}


void scan_keypad_rising_edge(){
    int r = 0;
    scan_keypad(key);
    _delay_ms(20);
    scan_keypad(pad);
    for(int i = 0; i < 16; i++){
        if(key[i]<pad[i]){            // endexomenws na ginei megalutero
            r = 1;
            rise[i] = 1;
        }
        else {
            rise[i] = 0;
        }

    }

}

uint8_t keypad_to_ascii(int a[]){
    for(int i = 0; i < 16; i++){

        if(a[i] == 1){

            return keypad[i];
        }
    }

    return 0;
}



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_PORT0 as output
uint8_t s = 0;

lcd_init();
lcd_data(0b00110000);
while(1){
```

```
scan_keypad_rising_edge();
s = keypad_to_ascii(rise);
if (s != 0){
  lcd_init();
  lcd_data(s);
}

}



}
```

Ζήτημα 6.2 :

Η λογική που ακολουθήσαμε φαίνεται στο παρακάτω διάγραμμα ροής.Να σημειωθεί ότι το σκανάρισμα του πληκτρολογίου έγινε με βάση τις συναρτήσεις που χρησιμοποιήθηκαν στην άσκηση 1.


Flag = 1
Δεν είναι λάθος
Διάβασε το πληκτρολόγιο S = Το πλήκτρο που διαβάστηκε
Διαβάστηκε πλήκτρο;
Όχι
Ναι
S = 3?
Όχι
Flag =0
Ναι
Διάβασε το πληκτρολόγιο S = Το πλήκτρο που διαβάστηκε
Διαβάστηκε πλήκτρο;
Όχι
Ναι
S = 9?
Όχι
Flag =0
Ναι
Flag = 1?
Όχι
Ναι
Άναψε τα λαμπακια για 4s Περίμενε αλλο 1s
Αναβόσβηνε τα λαμπάκια για 5s

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;
unsigned char keypad[16] = {0b00101010, 0b00110000, 0b00100011, 0b01000100, 0b00110111,
0b00111000, 0b00111001, 0b01000011, 0b00110100, 0b00110101, 0b00110110, 0b01000010,
0b00110001, 0b00110010, 0b00110011, 0b01000001};




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 key[16];
int pad[16];
int rise[16];

void  scan_row(int r, int a[], uint8_t c){
uint8_t y = 0x10;
```

```
        PCA9555_0_write(REG_OUTPUT_1, c);

    y =  ~PCA9555_0_read(REG_INPUT_1);

    a[r*4 + 0] = y & 0x10;
    a[r*4 + 1] = y & 0x20;
    a[r*4 + 2] = y & 0x40;
    a[r*4 + 3] = y & 0x80;

}



void scan_keypad(int a[]){
    scan_row(0,a,0xFE);
    scan_row(1,a,0xFD);
    scan_row(2,a,0xFB);
    scan_row(3,a,0xF7);
}

void scan_keypad_rising_edge(){
    int r = 0;
    scan_keypad(key);
    _delay_ms(10);
    scan_keypad(pad);
    for(int i = 0; i < 16; i++){
        if(key[i]>pad[i]){          // endexomenws na ginei megalutero
            r = 1;
            rise[i] = 1;
        }
        else {
            rise[i] = 0;
        }

    }

}

uint8_t keypad_to_ascii(int a[]){
    for(int i = 0; i < 16; i++){

        if(a[i] == 1){

            return keypad[i];
        }
    }

    return 0;
}



int main(){
    int f = 1;
  twi_init();



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

    uint8_t s = 0;
    DDRB |= 0xFF;
    f = 1;
    while(1){
        while(1){

        scan_keypad_rising_edge();
        s = keypad_to_ascii(rise);
        if (s != 0){
            if(s == 0b00110011 ) break;
            else {
                f = 0;
                break;
            }
        }

        }
        while(1){
          scan_keypad_rising_edge();
          s = keypad_to_ascii(rise);
          if (s != 0){
              if(s == 0b00111001 ) break;
              else {
                f = 0;
                break;
            }
        }
        }
        if(f == 1){
            PORTB = 0xFF;
            _delay_ms(4000);
            PORTB = 0x00;
            _delay_ms(1000);
        }
        else{
            for(int i = 0; i < 20; i++){
                if(i % 2 == 0){
                PORTB = 0xFF;
                }
                else{
                    PORTB = 0x00;
                }
                _delay_ms(250);

            }
        }
        f = 1;





    }
}
```