6η εργαστηριακή άσκηση

Φωτεινή Κυριακοπούλου 03120182
Ιωάννης Μπουφίδης 03120162

Ζήτημα 6.1

```
#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;
}

// read one byte from the twi device, read is followed by a stop condition
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;
	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)) return 1;
	TWDR0 = address;					// send device address
	TWCR0 = (1<<TWINT) | (1<<TWEN);
	// wail until transmission completed and ACK/NACK has been received
	while(!(TWCR0 & (1<<TWINT)));
	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;

		TWDR0 = address; // send device address
		TWCR0 = (1<<TWINT) | (1<<TWEN);

		while(!(TWCR0 & (1<<TWINT))); // wail 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);
```

```
                    // 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;
```

```
}

uint8_t scan_row(uint8_t row) {
    PCA9555_0_write(REG_OUTPUT_1, row); // set corresponding row to 0

    uint8_t keys = PCA9555_0_read(REG_INPUT_1);
    return keys; // return state of columns and rows
}

uint16_t scan_keypad(void) {
    uint16_t value = 0; // current state of the 16 buttons
    uint8_t keypad = 0;

    keypad = scan_row(0xFE); // check row 1
    if (keypad == 0xFE) value |= 0xF; // no button is pressed
    // find which button is pressed and move it to the appropriate position
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4; // leave space for the next check
    // repeat for remaining rows
    keypad = scan_row(0xFD); // check row 2
    if (keypad == 0xFD) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4;
    keypad = scan_row(0xFB); // check row 3
    if (keypad == 0xFB) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4;
    keypad = scan_row(0xF7); // check row 4
    if (keypad == 0xF7) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);

    return value;
}


uint16_t prev = 0xffff;
uint16_t scan_keypad_rising_edge() {
    // scan keypad twice with a delay of 20 ms
    uint16_t check1 = scan_keypad();
    _delay_ms(20);
    uint16_t check2 = scan_keypad();

    uint16_t final;
    // if second check is different return the most recent state
    if (check1 != check2) final = check2;
    else final = check1; // else the state of buttons hasn't changed

    // do not return the current state more than once
    if (final == prev) return 0xffff;
    prev = final;

    return final;
}
```

6η εργαστηριακή άσκηση

```
// convert pressed button to ascii
unsigned char keypad_to_ascii() {
    uint16_t x = scan_keypad_rising_edge(); // state of keypad
    unsigned char c = 0;
    switch (x) {// return corresponding character
        // 1st row
        case(0b1111111111111110): c='1'; break;
        case(0b1111111111111101): c='2'; break;
        case(0b1111111111111011): c='3'; break;
        case(0b1111111111110111): c='A'; break;
        // 2nd row
        case(0b1111111111101111): c='4'; break;
        case(0b1111111111011111): c='5'; break;
        case(0b1111111110111111): c='6'; break;
        case(0b1111111101111111): c='B'; break;
        // 3rd row
        case(0b1111111011111111): c='7'; break;
        case(0b1111110111111111): c='8'; break;
        case(0b1111101111111111): c='9'; break;
        case(0b1111011111111111): c='C'; break;
        // 4th row
        case(0b1110111111111111): c='*'; break;
        case(0b1101111111111111): c='0'; break;
        case(0b1011111111111111): c='#'; break;
        case(0b0111111111111111): c='D'; break;
    }
    return c;
}

int main(void) {
      unsigned char c;
      DDRB = 0xFF; // set PORTB as output
      twi_init();

      // IO1-IO3 (‘rows’) as outputs and IO4-IO7 (‘columns’) as inputs
      PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);

      while(1) {
            c = keypad_to_ascii(); //get corresponding character of the pressed button

            switch (c) {
            case '1': // Button "1" pressed
                  PORTB |= (1 << PB0); // Turn on LED at PB0
                  break;
            case '5': // Button "5" pressed
                  PORTB |= (1 << PB1); // Turn on LED at PB1
                  break;
            case '9': // Button "9" pressed
                  PORTB |= (1 << PB2); // Turn on LED at PB2
                  break;
            case 'D': // Button "D" pressed
                  PORTB |= (1 << PB3); // Turn on LED at PB3
```

```
                        break;
                default:
                        PORTB = 0x00;
                }
        _delay_ms(200);
        }
        return 0;
}
```

Ζήτημα 6.2

```
#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;
}

// read one byte from the twi device, read is followed by a stop condition
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;
	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)) return 1;
	TWDR0 = address;					// send device address
	TWCR0 = (1<<TWINT) | (1<<TWEN);
	// wail until transmission completed and ACK/NACK has been received
	while(!(TWCR0 & (1<<TWINT)));
	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;

		TWDR0 = address; // send device address
		TWCR0 = (1<<TWINT) | (1<<TWEN);

		while(!(TWCR0 & (1<<TWINT))); // wail 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);

			// 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;
}

uint8_t scan_row(uint8_t row) {
```

```
    PCA9555_0_write(REG_OUTPUT_1, row); // set corresponding row to 0

    uint8_t keys = PCA9555_0_read(REG_INPUT_1);
    return keys; // return state of columns and rows
}

uint16_t scan_keypad(void) {
    uint16_t value = 0; // current state of the 16 buttons
    uint8_t keypad = 0;

    keypad = scan_row(0xFE); // check row 1
    if (keypad == 0xFE) value |= 0xF; // no button is pressed
    // find which button is pressed and move it to the appropriate position
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4; // leave space for the next check
    // repeat for remaining rows
    keypad = scan_row(0xFD); // check row 2
    if (keypad == 0xFD) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4;
    keypad = scan_row(0xFB); // check row 3
    if (keypad == 0xFB) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4;
    keypad = scan_row(0xF7); // check row 4
    if (keypad == 0xF7) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);

    return value;
}


uint16_t prev = 0xffff;
uint16_t scan_keypad_rising_edge() {
    // scan keypad twice with a delay of 20 ms
    uint16_t check1 = scan_keypad();
    _delay_ms(20);
    uint16_t check2 = scan_keypad();

    uint16_t final;
    // if second check is different return the most recent state
    if (check1 != check2) final = check2;
    else final = check1; // else the state of buttons hasn't changed

    // do not return the current state more than once
    if (final == prev) return 0xffff;
    prev = final;

    return final;
}

// convert pressed button to ascii
unsigned char keypad_to_ascii() {
```

```
    uint16_t x = scan_keypad_rising_edge(); // state of keypad
    unsigned char c = 0;
    switch (x) {// return corresponding character
        // 1st row
        case(0b1111111111111110): c='1'; break;
        case(0b1111111111111101): c='2'; break;
        case(0b1111111111111011): c='3'; break;
        case(0b1111111111110111): c='A'; break;
        // 2nd row
        case(0b1111111111101111): c='4'; break;
        case(0b1111111111011111): c='5'; break;
        case(0b1111111110111111): c='6'; break;
        case(0b1111111101111111): c='B'; break;
        // 3rd row
        case(0b1111111011111111): c='7'; break;
        case(0b1111110111111111): c='8'; break;
        case(0b1111101111111111): c='9'; break;
        case(0b1111011111111111): c='C'; break;
        // 4th row
        case(0b1110111111111111): c='*'; break;
        case(0b1101111111111111): c='0'; break;
        case(0b1011111111111111): c='#'; break;
        case(0b0111111111111111): c='D'; break;
    }
    return c;
}

// swap macro
uint8_t swap(uint8_t x) {
    return (uint8_t)((x >> 4) | (x << 4));
}

void write_2_nibbles(uint8_t command) {
    // write MSB nibble
    PORTD = ((PIND & 0X0f) | (command & 0xf0));

    PORTD |= (1 << PD3);  // Enable pulse
    _delay_us(1);
    PORTD &= ~(1 << PD3); // Disable pulse

    // write LSB nibble
    PORTD = ((PIND & 0X0f) | (swap(command) & 0xf0));

    PORTD |= (1 << PD3);
    _delay_us(1);
    PORTD &= ~(1 << PD3);
}

// send a command to lcd's controller (4-bit mode)
void lcd_command(uint8_t command) {
    PORTD &= ~(1 << PD2);  // LCD_RS = 0 => instruction
    write_2_nibbles(command);  // send instruction
    _delay_us(250);  // wait 250us
```

```
}

// send data to lcd's controller (4-bit mode)
void lcd_data(uint8_t data) {
    PORTD |= (1 << PD2);  // LCD_RS = 1 => data
    write_2_nibbles(data);  // send instruction
    _delay_us(250);  // wait 250us
}

void lcd_clear_display() {
    lcd_command(0x01);
    _delay_ms(5);
}

// lcd monitor set-up
void lcd_init() {
    _delay_ms(200);  // wait 200ms

    // set microcontroler in 4-bit mode

    for (int i=0; i<3; i++) {  // 3 times
        PORTD = 0x30;  // switch to 8-bit mode
        PORTD |= (1 << PD3);  // Enable pulse
        _delay_us(1);
        PORTD &= ~(1 << PD3); // Disable pulse
        _delay_us(250);
    }

    PORTD = 0x20;  // switch to 4-bit mode
    PORTD |= (1 << PD3);  // Enable pulse
    _delay_us(1);
    PORTD &= ~(1 << PD3); // Disable the enable pulse
    _delay_us(250);

    lcd_command(0x28);   // 5x8 dots, 2 lines
    lcd_command(0x0c);   // display on, cursor-blinking off
    lcd_clear_display(); // clear display
    lcd_command(0x06);   // increase address, no display shift
}


int main() {
      unsigned char c;
      prev = 0xffff;

      // initialize PORTD for LCD
      DDRD = 0xff;
      twi_init();
      PCA9555_0_write(REG_CONFIGURATION_1, 0xf0);

      lcd_init(); // initialize LCD screen
      _delay_ms(100);
```

```
	while(1) {
		c = keypad_to_ascii(); // find if a key is pressed

		if (c != 0) {
			lcd_clear_display(); // clear display
			lcd_data(c); // display key
		}
	}

	return 0;
}
```

Ζήτημα 6.3

```
#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;
}

// read one byte from the twi device, read is followed by a stop condition
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;
	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)) return 1;
	TWDR0 = address;					// send device address
	TWCR0 = (1<<TWINT) | (1<<TWEN);
	// wail until transmission completed and ACK/NACK has been received
	while(!(TWCR0 & (1<<TWINT)));
	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;

		TWDR0 = address; // send device address
		TWCR0 = (1<<TWINT) | (1<<TWEN);

		while(!(TWCR0 & (1<<TWINT))); // wail 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);

			// 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;
}

uint8_t scan_row(uint8_t row) {
```

```
    PCA9555_0_write(REG_OUTPUT_1, row); // set corresponding row to 0

    uint8_t keys = PCA9555_0_read(REG_INPUT_1);
    return keys; // return state of columns and rows
}

uint16_t scan_keypad(void) {
    uint16_t value = 0; // current state of the 16 buttons
    uint8_t keypad = 0;

    keypad = scan_row(0xFE); // check row 1
    if (keypad == 0xFE) value |= 0xF; // no button is pressed
    // find which button is pressed and move it to the appropriate position
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4; // leave space for the next check
    // repeat for remaining rows
    keypad = scan_row(0xFD); // check row 2
    if (keypad == 0xFD) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4;
    keypad = scan_row(0xFB); // check row 3
    if (keypad == 0xFB) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);
    value <<= 4;
    keypad = scan_row(0xF7); // check row 4
    if (keypad == 0xF7) value |= 0xF;
    else value |= ((keypad & 0xF0) >> 4);

    return value;
}


uint16_t prev = 0xffff;
uint16_t scan_keypad_rising_edge() {
    // scan keypad twice with a delay of 20 ms
    uint16_t check1 = scan_keypad();
    _delay_ms(20);
    uint16_t check2 = scan_keypad();

    uint16_t final;
    // if second check is different return the most recent state
    if (check1 != check2) final = check2;
    else final = check1; // else the state of buttons hasn't changed

    // do not return the current state more than once
    if (final == prev) return 0xffff;
    prev = final;

    return final;
}

// convert pressed button to ascii
unsigned char keypad_to_ascii() {
```

```
    uint16_t x = scan_keypad_rising_edge(); // state of keypad
    unsigned char c = 0;
    switch (x) {// return corresponding character
        // 1st row
        case(0b1111111111111110): c='1'; break;
        case(0b1111111111111101): c='2'; break;
        case(0b1111111111111011): c='3'; break;
        case(0b1111111111110111): c='A'; break;
        // 2nd row
        case(0b1111111111101111): c='4'; break;
        case(0b1111111111011111): c='5'; break;
        case(0b1111111110111111): c='6'; break;
        case(0b1111111101111111): c='B'; break;
        // 3rd row
        case(0b1111111011111111): c='7'; break;
        case(0b1111110111111111): c='8'; break;
        case(0b1111101111111111): c='9'; break;
        case(0b1111011111111111): c='C'; break;
        // 4th row
        case(0b1110111111111111): c='*'; break;
        case(0b1101111111111111): c='0'; break;
        case(0b1011111111111111): c='#'; break;
        case(0b0111111111111111): c='D'; break;
    }
    return c;
}

int main(void) {
      unsigned char character1 = 0, character2 = 0;
      prev = 0xffff;
      DDRB |= 0b00111111; // set PB0-PB5 as output
      PORTB = 0x00; // turn of LEDs

      twi_init();
      PCA9555_0_write(REG_CONFIGURATION_1, 0xf0);

      while(1) {
            while(character1 == 0) { // wait for first digit
                  character1 = keypad_to_ascii();
            }
            _delay_ms(5);
            while(character2 == 0) { // wait for second digit
                  character2 = keypad_to_ascii();
            }

            if (character1 == '1' && character2 == '1') { // code is correct
                  PORTB = 0b00111111; // turn on LEDs for 4 seconds
                  _delay_ms(4000);
                  PORTB = 0x00;
            }
            else {
                  for(int i = 0; i < 10; i++) { // code is incorrect
                        PORTB = 0b00111111;  // blinking of LEDs every 250 ms
```

```
                        _delay_ms(250);       // for a total of 5 seconds
                        PORTB = 0x00;
                        _delay_ms(250);
                    }
                }

                character1 = 0;
                character2 = 0;
                _delay_ms(5000); // wait 5 seconds before next input
            }
            return 0;
}
```

Το **διάγραμμα ροής** του παραπάνω προγράμματος είναι το εξής: