5η εργαστηριακή άσκηση

Φωτεινή Κυριακοπούλου 03120182
Ιωάννης Μπουφίδης 03120162

Ζήτημα 5.1

```
#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;
```

```
}


int main(void) {
      uint8_t val, A, B, C, D;
      uint8_t f0, f1;
      twi_init();
      DDRB &= ~(0x0F);  // set bits 0 to 3 of PORTB as input
      PCA9555_0_write(REG_CONFIGURATION_0, 0x00); // set EXT_PORT0 as output

      while(1) {
            val = PINB; // read input
            val = ~val; // reverse logic
            A = (val >> 0) & 0x01;  // extract the first bit
            B = (val >> 1) & 0x01;  // extract the second bit
            C = (val >> 2) & 0x01;  // extract the third bit
            D = (val >> 3) & 0x01;  // extract the fourth bit

            f0 = ~((~A  & B) | (~B & C & D)) & 0x01; // compute f0
            f1 = ((A & C) & (B | D)) << 1;  // compute f1 and shift it left in the
            place of the 2nd bit

            PCA9555_0_write(REG_OUTPUT_0, f0 | f1);
            // combine f0 and f1 and write result
      }
}
```

Ζήτημα 5.2

```
#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)));
```

```c
	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));
```

```c
                        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;
}
```

```c
int main(void) {
      twi_init();
      // set IO0_0-3 as output
      PCA9555_0_write(REG_CONFIGURATION_0, 0xf0);
      // set IO1_0-3 as output and IO1_4-7 as input
      PCA9555_0_write(REG_CONFIGURATION_1, 0xf0);
      // set IO1_0 "On"
      PCA9555_0_write(REG_OUTPUT_1, 0b1110);
      // initialize-turn off 4 MSB LEDs of PORTD
      DDRD=0xFF; PORTD=0;

      uint8_t keys;
      while(1) {
            // input from IO1_4-7
            keys = ~(PCA9555_0_read(REG_INPUT_1));
            // display on LEDs accordingly
            if (keys != 0) {
                  if ((keys & (1<<4)) != 0) PCA9555_0_write(REG_OUTPUT_0, 0x01);
                  if ((keys & (1<<5)) != 0) PCA9555_0_write(REG_OUTPUT_0, 0x02);
                  if ((keys & (1<<6)) != 0) PCA9555_0_write(REG_OUTPUT_0, 0x04);
                  if ((keys & (1<<7)) != 0) PCA9555_0_write(REG_OUTPUT_0, 0x08);
                  PCA9555_0_write(REG_OUTPUT_0, 0x00);
            }
            else PCA9555_0_write(REG_OUTPUT_0, 0x00);
      }
}
```