Εργαστήριο Μικρουπολογιστών

Παπαδόπουλος Κωνσταντίνος el20152
Σκούρτης Παύλος el20052

Αρχείο communication.h με τις συναρτήσεις της διεπαφής

```
#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;
```

```
// 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;
}
```

Άσκηση 3

#define F_CPU 16000000UL	
#include<avr/io.h>	
#include<avr/interrupt.h>	
#include<util/delay.h>	
#include<comunication.h>	Συναρτήσεις διεπαφής
int main(void) {	
twi_init();	
PCA9555_0_write(REG_CONFIGURATION_0, 0x00);	//Set EXT_PORT0 as output
DDRB=0b00000000;	PORTD—>input
while(1)	
{	
int output,input,A,B,C,D,comA,comB,F0,F1;	
input=PINB;	
A=input&0b00000001;	Υπολογισμός λογικής συνάντησης και αποθήκευση στο output
comA=~A;	
comA=comA&0b00000001;	
comB=~B;	
comB=comB&0b00000001;	
B=input&0b00000010;	
B=B>>1;	
C=input&0b00000100;	
C=C>>2;	
D=input&0b00001000;	
D=D>>3;	
F0=(comA&B)|(comB&C&D);	
F0=~F0;	

F0=F0&0b00000001;	
F1=(A&C)&(B|D);	
F1=F1<<1;	
output=F0|F1;	
PCA9555_0_write(REG_OUTPUT_0, output)	Έξοδος στη θύρα επέκτασης 0
}	
}	

ΑΣΚΗΣΗ 1

#define F_CPU 16000000UL	
#include<avr/io.h>	
#include<avr/interrupt.h>	
#include<util/delay.h>	
#include<comunication.h>	Συναρτήσεις διεπαφής
int main(void) {	
twi_init();	
PCA9555_0_write(REG_CONFIGURATION_0, 0x00);	**//Set EXT_PORT0 as output**
PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);	**//Set for EXT_PORT1 P4-P7 as input and P0(also P1-P3) as output**

PCA9555_0_write(REG_OUTPUT_1, 0x00);	**//Set IO1 as to enable pull up to IO1_0 στην θυρα επέκτασης 1**
while(1)	
{	
uint8_t read;	
read=PCA9555_0_read(REG_INPUT_1);	
read=~read;	Παίρνω το συμπλήρωμα λόγω αρνητικής λογικής από στην είσοδο
read=read>>4;	Κάνω shift 4 θέσεις αφού με ενδιαφέρουν μόνο τα 4 msb στις θέσεις των 4 lsb
PCA9555_0_write(REG_OUTPUT_0, read);	Έξοδος στη θύρα επέκτασης 0
}	
}	