Χρήση εξωτερικών Θυρών Επέκτασης στον AVR

Εργαστήριο Μικροϋπολογιστών

Κριθαρίδης Κωνσταντίνος, *el*21045 Μπαλάτος Δημήτριος, *el*21170

12 Νοεμβρίου 2024

1 Ζήτημα 5.1

Αρχικά, μεταφέρουμε τον κώδικα της εκφώνησης για την επικοινωνία με το PCA9555 μέσω της διεπαφής TWI. Στην συνάρτηση main, αφού αρχικοποιήσουμε το TWI, και θέσουμε το `EXT_PORT0` ως έξοδο μέσω του καταχωρητή `REG_CONFIGURATION_0` και το PORTB ως input, ξεκινάμε να διαβάζουμε διαρκώς το PINB (το οποίο μετατρέπουμε σε θετική λογική). Αποθηκεύουμε σε διαφορετικές μεταβλητές τα A, B, C, D και υπολογίζουμε τα $F0$, $F1$ με βάση αυτές. Τέλος, γράφουμε στο `EXT_PORT0` μέσω του καταχωρητή `REG_OUTPUT_0` τις τιμές των $F0$, $F1$. Αφού έχουμε συνδέσει τους ακροδέκτες `IO0_0` και `IO0_1` του κονέκτορα `P18` με τους ακροδέκτες `LED_PD2` και `LED_PD3` , βλέπουμε το αποτέλεσμα που γράφουμε στο `EXT_PORT0` στα αντίστοιχα LEDs.

```c
/*
 * main.c
 *
 * Created: 11/8/2024 10:58:48 AM
 *  Author: User
 */

#include <xc.h>

#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)
{
```

```
78          uint8_t twi_status;
79
80          // send START condition
81          TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
82
83          // wait until transmission completed
84          while(!(TWCR0 & (1<<TWINT)));
85
86          // check value of TWI Status Register.
87          twi_status = TW_STATUS & 0xF8;
88          if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
89
90          // send device address
91          TWDR0 = address;
92          TWCR0 = (1<<TWINT) | (1<<TWEN);
93
94          // wail until transmission completed and ACK/NACK has been received
95          while(!(TWCR0 & (1<<TWINT)));
96          // check value of TWI Status Register.
97          twi_status = TW_STATUS & 0xF8;
98          if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
99          {
100                 return 1;
101         }
102         return 0;
103 }
104
105 // Send start condition, address, transfer direction.
106 // Use ack polling to wait until device is ready
107 void twi_start_wait(unsigned char address)
108 {
109         uint8_t twi_status;
110         while ( 1 )
111         {
112                 // send START condition
113                 TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
114
115                 // wait until transmission completed
116                 while(!(TWCR0 & (1<<TWINT)));
117
118                 // check value of TWI Status Register.
119                 twi_status = TW_STATUS & 0xF8;
120                 if ( (twi_status != TW_START) && (twi_status != TW_REP_START))
                    ↪  continue;
121
122                 // send device address
123                 TWDR0 = address;
124                 TWCR0 = (1<<TWINT) | (1<<TWEN);
125
126                 // wail until transmission completed
127                 while(!(TWCR0 & (1<<TWINT)));
```

```
128
129                 // check value of TWI Status Register.
130                 twi_status = TW_STATUS & 0xF8;
131                 if ( (twi_status == TW_MT_SLA_NACK )||(twi_status
                     ↪  ==TW_MR_DATA_NACK) )
132                 {
133                         /* device busy, send stop condition to terminate write
                             ↪  operation */
134                         TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
135
136                         // wait until stop condition is executed and bus released
137                         while(TWCR0 & (1<<TWSTO));
138                         continue;
139                 }
140                 break;
141         }
142 }
143
144 // Send one byte to twi device, Return 0 if write successful or 1 if write failed
145 unsigned char twi_write( unsigned char data )
146 {
147         // send data to the previously addressed device
148         TWDR0 = data;
149         TWCR0 = (1<<TWINT) | (1<<TWEN);
150
151         // wait until transmission completed
152         while(!(TWCR0 & (1<<TWINT)));
153         if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
154         return 0;
155 }
156
157 // Send repeated start condition, address, transfer direction
158 //Return: 0 device accessible
159 // 1 failed to access device
160 unsigned char twi_rep_start(unsigned char address)
161 {
162         return twi_start( address );
163 }
164
165 // Terminates the data transfer and releases the twi bus
166 void twi_stop(void)
167 {
168         // send stop condition
169         TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
170
171         // wait until stop condition is executed and bus released
172         while(TWCR0 & (1<<TWSTO));
173 }
174
175 void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
176 {
```

```
        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;
}

#include <stdbool.h>

int main(void) {
        twi_init();
        PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output

        DDRB = 0x00; //Set PORTB as input

        bool A, B, C, D, F0, F1;

        while(1)
        {
                uint8_t pins = ~PINB;
                A = pins&(1<<0); B = pins&(1<<1); C = pins&(1<<2); D =
 ↪  pins&(1<<3);
                F0 = !((!A && B && C) || (!B && D));
                F1 = (A || B || C) && (B && !D);

                PCA9555_0_write(REG_OUTPUT_0, F0|(F1<<1));
        }
}
```

2 Ζήτημα 5.2

Μεταφέρουμε τον κώδικα της εκφώνησης για την επικοινωνία με το PCA9555, όπως και παραπάνω. Για την χρήση του numpad, στέλνουμε High Voltage στα pins που αντιστοιχούν στις πάνω 3 γραμμές και Low Voltage στην γραμμή που θέλουμε να ενεργοποιήσουμε. Στην συνέχεια διαβάζουμε συνεχώς το `IO1` και κρατάμε τα bits που αντιστοιχούν στις στήλες· με έναν βρόχο βρίσκουμε ποια στήλη είναι πατημένη (Low Voltage) και ενημερώνουμε αντίστοιχα τα LEDs του `J18` . Σημ.: για απλοποίηση του κώδικα χρησιμοποιήσαμε την συμπληρωματική ως προς 1 τιμή του `REG_INPUT_1` .

```
/*
 * main.c
 *
```

```
 * Created: 11/8/2024 10:58:48 AM
 *  Author: User
 */

#include <xc.h>

#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
```

```
55          TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
56  }
57
58  // Read one byte from the twi device (request more data from device)
59  unsigned char twi_readAck(void)
60  {
61          TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
62          while(!(TWCR0 & (1<<TWINT)));
63          return TWDR0;
64  }
65
66  //Read one byte from the twi device, read is followed by a stop condition
67  unsigned char twi_readNak(void)
68  {
69          TWCR0 = (1<<TWINT) | (1<<TWEN);
70          while(!(TWCR0 & (1<<TWINT)));
71          return TWDR0;
72  }
73
74  // Issues a start condition and sends address and transfer direction.
75  // return 0 = device accessible, 1= failed to access device
76  unsigned char twi_start(unsigned char address)
77  {
78          uint8_t twi_status;
79
80          // send START condition
81          TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
82
83          // wait until transmission completed
84          while(!(TWCR0 & (1<<TWINT)));
85
86          // check value of TWI Status Register.
87          twi_status = TW_STATUS & 0xF8;
88          if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
89
90          // send device address
91          TWDR0 = address;
92          TWCR0 = (1<<TWINT) | (1<<TWEN);
93
94          // wail until transmission completed and ACK/NACK has been received
95          while(!(TWCR0 & (1<<TWINT)));
96          // check value of TWI Status Register.
97          twi_status = TW_STATUS & 0xF8;
98          if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
99          {
100                 return 1;
101         }
102         return 0;
103 }
104
105 // Send start condition, address, transfer direction.
```

```
106 // Use ack polling to wait until device is ready
107 void twi_start_wait(unsigned char address)
108 {
109         uint8_t twi_status;
110         while ( 1 )
111         {
112                 // send START condition
113                 TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
114 
115                 // wait until transmission completed
116                 while(!(TWCR0 & (1<<TWINT)));
117 
118                 // check value of TWI Status Register.
119                 twi_status = TW_STATUS & 0xF8;
120                 if ( (twi_status != TW_START) && (twi_status != TW_REP_START))
                     ↪  continue;
121 
122                 // send device address
123                 TWDR0 = address;
124                 TWCR0 = (1<<TWINT) | (1<<TWEN);
125 
126                 // wail until transmission completed
127                 while(!(TWCR0 & (1<<TWINT)));
128 
129                 // check value of TWI Status Register.
130                 twi_status = TW_STATUS & 0xF8;
131                 if ( (twi_status == TW_MT_SLA_NACK )||(twi_status
                     ↪  ==TW_MR_DATA_NACK) )
132                 {
133                         /* device busy, send stop condition to terminate write
                             ↪  operation */
134                         TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
135 
136                         // wait until stop condition is executed and bus released
137                         while(TWCR0 & (1<<TWSTO));
138                         continue;
139                 }
140                 break;
141         }
142 }
143 
144 // Send one byte to twi device, Return 0 if write successful or 1 if write failed
145 unsigned char twi_write( unsigned char data )
146 {
147         // send data to the previously addressed device
148         TWDR0 = data;
149         TWCR0 = (1<<TWINT) | (1<<TWEN);
150 
151         // wait until transmission completed
152         while(!(TWCR0 & (1<<TWINT)));
153         if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
```

```
154         return 0;
155 }
156
157 // Send repeated start condition, address, transfer direction
158 //Return: 0 device accessible
159 // 1 failed to access device
160 unsigned char twi_rep_start(unsigned char address)
161 {
162         return twi_start( address );
163 }
164
165 // Terminates the data transfer and releases the twi bus
166 void twi_stop(void)
167 {
168         // send stop condition
169         TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
170
171         // wait until stop condition is executed and bus released
172         while(TWCR0 & (1<<TWSTO));
173 }
174
175 void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
176 {
177         twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
178         twi_write(reg);
179         twi_write(value);
180         twi_stop();
181 }
182
183 uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
184 {
185         uint8_t ret_val;
186         twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
187         twi_write(reg);
188         twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
189         ret_val = twi_readNak();
190         twi_stop();
191         return ret_val;
192 }
193
194 int main(void) {
195         twi_init();
196         PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
197         PCA9555_0_write(REG_CONFIGURATION_1, 0xf0); //Set EXT_PORT1 as: 0:3 ->
 ↪  output
198                                                     //                4:7 -> input
199
200         // initialize numpad rows
201         PCA9555_0_write(REG_OUTPUT_0, 0x00);
202         PCA9555_0_write(REG_OUTPUT_1, 0x0e);
203
```

```
        while(1)
        {
                unsigned int in = ~PCA9555_0_read(REG_INPUT_1);
                in >>= 4; // remove IO1[0:3]
                unsigned int out = in ? 1 : 0;
                while (!(in & 1)) {
                        out <<= 1; in >>= 1;
                }
                PCA9555_0_write(REG_OUTPUT_0, out);
        }
}
```

3 Ζήτημα 5.3

Μεταφέρουμε τον κώδικα, όπως και παραπάνω, καθώς και τον κώδικα επικοινωνίας με την LCD σε C από την σειρά 4. Αλλάζουμε τις εγγραφές στο `PORTD` με συναρτήσης PCA9555, ενώ για να γλιτώσουμε μερικά reads ορίζουμε global μεταβλητή `LAST` που ενημερώνουμε κατά τις εγγραφές. Για διευκόλυνση, ορίζουμε συνάρτηση `flash()` που τρέχει συχνά επαναλαμβανόμενές εντολές κατά την χρήση της LCD, καθώς και `lcd_string(str)` για την εκτύπωση c string στην οθόνη (σε περίπτωση επιλογής `'\n'`, τοποθετούμε τον κέρσορα στην πρώτη στήλη της δεύτερης γραμμής).

```
/*
 * main.c
 *
 * Created: 11/8/2024 10:58:48 AM
 *  Author: User
 */

#include <xc.h>

#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

#define NOP() do { __asm__ __volatile__ ( "nop "); } while (0)

// 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));
}

uint8_t LAST;

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
{
```

```
181         twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
182         twi_write(reg);
183         twi_write(value);
184         twi_stop();
185         LAST = value;
186         //if (reg != REG_CONFIGURATION_0) exit(0);
187 }
188
189 uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
190 {
191         uint8_t ret_val;
192         twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
193         twi_write(reg);
194         twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
195         ret_val = twi_readNak();
196         twi_stop();
197         return ret_val;
198 }
199
200 void flash ()
201 {
202         _delay_us(50);
203         uint8_t tmp = PCA9555_0_read(REG_INPUT_0);
204         PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 3));
205         _delay_us(50);
206         PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 3));
207 }
208
209 void write_2_nibbles(uint8_t data){
210         uint8_t temp = LAST & 0x0f;
211         uint8_t out = data & 0xf0 | temp;
212         PCA9555_0_write(REG_OUTPUT_0, out);
213         flash();
214
215         out = (data << 4) & 0xf0 | temp;
216         PCA9555_0_write(REG_OUTPUT_0, out);
217         flash();
218 }
219
220 void lcd_data (uint8_t data)
221 {
222         uint8_t tmp = LAST;
223         PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 2));
224         write_2_nibbles(data);
225         _delay_us(500);
226 }
227
228 void lcd_command (uint8_t instr)
229 {
230         uint8_t tmp = LAST;
231         PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 2));
```

```
232         write_2_nibbles(instr);
233         _delay_us(500);
234 }
235
236 void lcd_clear_display(){
237         lcd_command(0x01);
238         _delay_ms(200);
239 }
240
241 void lcd_init ()
242 {
243         _delay_ms(200);
244
245         uint8_t out = 0x30;
246         for (int i=0; i<3; ++i) {
247                 PCA9555_0_write(REG_OUTPUT_0, out);
248                 flash();
249                 _delay_us(250);
250         }
251         PCA9555_0_write(REG_OUTPUT_0, 0x20);
252         flash();
253         _delay_us(250);
254
255         lcd_command(0x28);
256         lcd_command(0x0c);
257         lcd_clear_display();
258         lcd_command(0x06);
259 }
260
261 void lcd_string (const char* str)
262 {
263         lcd_clear_display();
264         for (; *str; str++) {
265                 if (*str == '\n')
266                         lcd_command(0xc0);
267                 else
268                         lcd_data(*str);
269         }
270 }
271
272 const char name[] = "Jim Balatos\nKon/nos Krith.";
273
274 int main(void) {
275         DDRB = 0xff;
276         twi_init();
277         PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
278         lcd_init();
279         lcd_string(name);
280         while (1) {}
281 }
```