Χρήση εξωτερικών Θυρών Επέκτασης στον AVR

Εργαστήριο Μικροϋπολογιστών

Κριθαρίδης Κωνσταντίνος, $el21045$ Μπαλάτος Δημήτριος, $el21170$

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)
{
```

```c
        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;
}

#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`.

```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)
{
        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;
```

```c
        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) {
        twi_init();
        PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
        PCA9555_0_write(REG_CONFIGURATION_1, 0xf0); //Set EXT_PORT1 as: 0:3 -> output
                                                    //                4:7 -> input

        // initialize numpad rows
        PCA9555_0_write(REG_OUTPUT_0, 0x00);
        PCA9555_0_write(REG_OUTPUT_1, 0x0e);

        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,
```

```c
        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)
{
```

```c
        twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
        twi_write(reg);
        twi_write(value);
        twi_stop();
        LAST = value;
        //if (reg != REG_CONFIGURATION_0) exit(0);
}

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 flash ()
{
        _delay_us(50);
        uint8_t tmp = PCA9555_0_read(REG_INPUT_0);
        PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 3));
        _delay_us(50);
        PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 3));
}

void write_2_nibbles(uint8_t data){
        uint8_t temp = LAST & 0x0f;
        uint8_t out = data & 0xf0 | temp;
        PCA9555_0_write(REG_OUTPUT_0, out);
        flash();

        out = (data << 4) & 0xf0 | temp;
        PCA9555_0_write(REG_OUTPUT_0, out);
        flash();
}

void lcd_data (uint8_t data)
{
        uint8_t tmp = LAST;
        PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 2));
        write_2_nibbles(data);
        _delay_us(500);
}

void lcd_command (uint8_t instr)
{
        uint8_t tmp = LAST;
        PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 2));
        write_2_nibbles(instr);
        _delay_us(500);
}

void lcd_clear_display(){
        lcd_command(0x01);
        _delay_ms(200);
}

void lcd_init ()
{
        _delay_ms(200);

        uint8_t out = 0x30;
        for (int i=0; i<3; ++i) {
                PCA9555_0_write(REG_OUTPUT_0, out);
                flash();
                _delay_us(250);
        }
        PCA9555_0_write(REG_OUTPUT_0, 0x20);
        flash();
        _delay_us(250);

        lcd_command(0x28);
        lcd_command(0x0c);
        lcd_clear_display();
        lcd_command(0x06);
}

void lcd_string (const char* str)
{
        lcd_clear_display();
        for (; *str; str++) {
                if (*str == '\n')
                        lcd_command(0xc0);
                else
                        lcd_data(*str);
        }
}

const char name[] = "Jim Balatos\nKon/nos Krith.";

int main(void) {
        DDRB = 0xff;
        twi_init();
        PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
        lcd_init();
        lcd_string(name);
        while (1) {}
}
```