Χρήση Οθόνης 2 × 16 Χαρακτήρων στον AVR

Εργαστήριο Μικροϋπολογιστών

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

7 Νοεμβρίου 2024

1 Ζήτημα 4.1

Για την άσκηση, διαβάζουμε ανά το ζητούμενο διάστημα την αναλογική είσοδο. Από εδώ και πέρα, αναγκαζόμαστε να κάνουμε πράξεις με αριθμούς μήκους πάνω από 8 bits· αυτές τις υλοποιούμε μόνοι μας χωρίς ρουτίνες, καθώς απαιτούνται μόνο μία φορά. Για να απλοποιήσουμε την διαίρεση, πολλαπλασιάζουμε πρώτα με το 500, έτσι ώστε η διαίρεση να είναι με το 1024 και να απαιτεί μόνο ολισθήσεις. Για την εκτύπωση, κάνουμε διαδοχικές διαιρέσεις με το 10 και σπρώχνουμε τα υπόλοιπα στο stack (έτσι ώστε να έχουμε σωστή σειρά ψηφίων στην ανάγνωση μετά). Τέλος, εκτυπώνουμε τα ψηφία και εκτυπώνουμε την υποδιαστολή μετά το πρώτο.

```
;
; Ex4_1.asm
;
; Created: 11/1/2024 11:47:20 AM
; Author : User
;

.include "m328PBdef.inc"

.org 0x0
        rjmp reset
.org 0x2a
        rjmp adcisr

.def tmp  = r16
.def lcd  = r17
.def val0 = r18
.def val1 = r19
.def tmp0 = r20
.def tmp1 = r21
.def i    = r22
.def k    = r23
.def argl = r24
.def argh = r25
.def res0 = r26
.def res1 = r27
.def res2 = r28
.def res3 = r29

```

```
.equ PD2 = 2
.equ PD3 = 3

adcisr:
        push tmp                ;
        in tmp, SREG                ; Push into stack
        push tmp                ;

        lds val0, ADCL                ;
        lds val1, ADCH                ; Get analog input
        ; out PORTB, val0

        pop tmp                ;
        out SREG, tmp        ; Pop from stack
        pop tmp                ;
        reti
        /*; Multiply x500{{{
        ldi tmp0, LOW(500)
        ldi tmp1, HIGH(500)

        clr tmp

        mul val0, tmp0
        movw res0, r0
        clr res2
        clr res3

        mul val0, tmp1
        add res1, r0
        adc res2, r1
        adc res3, tmp

        mul val1, tmp0
        add res1, r0
        adc res2, r1
        adc res3, tmp

        mul val1, tmp1
        add res2, r0
        adc res3, r1; }}}

        ; Divide by 1024
        ldi i, 10
_loop:        clc
        ror res2 ror res1 ror res0
        dec i
        brne _loop
        ; HAVE THE ANSWER, 4 digits (1 + comma + 2 decimals)

        ldi i,3
        ; Convert into decimals
```

```
convert:
        rcall divide_by10 ; Quotient in res0:1, remainder in r0:1
        mov tmp, r0
        subi tmp, -0x30         ; Convert into char
        push tmp                 ; Push to stack
        dec i
        brne convert

        ; Output
        rcall lcd_clear_display
        pop argl rcall lcd_data                 ; First digit
        ldi argl, 0x2e rcall lcd_data        ; Period
        pop argl rcall lcd_data                 ; First decimal
        pop argl rcall lcd_data                 ; Second decimal
*/

reset:
        ldi tmp, high(RAMEND)         ;
        out SPH, tmp                 ; Initialize SP
        ldi tmp, low(RAMEND)         ;
        out SPL, tmp                 ;

        ser tmp                         ; Set PORTD (LSD) as output
        out DDRD, tmp                 ;
        out DDRB, tmp

        ; ser tmp out PORTB, tmp
        ; rcall lcd_init                 ; Initialize LCD
        ; rcall lcd_clear_display
        ; clr tmp out PORTB, tmp

        ldi tmp, 0b01000001         ; Set ADC1: 5V, Right aligned, ADC1
        sts ADMUX, tmp                 ;
        ldi tmp, 0b10001111         ; Enable, Interrupts, 125kHz
        sts ADCSRA, tmp

        sei                         ; Enable interrupts
        ldi k, 0

start:
        /*lds tmp, ADCSRA
        ori tmp, 1 << ADSC
        sts ADCSRA, tmp*/

        inc k
        out PORTB, k

        ldi r24, LOW(100)         ; Set delay time to 100ms
        ldi r25, HIGH(100)         ;
        rcall wait_msec                 ; Loop with delay
        rjmp start
```

```
132
133 divide_by10:
134         push tmp                ;
135         in tmp, SREG                ; Push into stack
136         push tmp                ;
137         push argl push argh
138
139         clr argl clr argh
140 div_loop:
141         tst res1
142         brne div_cont
143         cpi res0, 10
144         brlt div_done
145 div_cont:
146         sbiw res0, 10
147         adiw argl, 1
148         rjmp div_loop
149 div_done:
150         mov r0, res0 mov res0, argl
151         mov r1, res1 mov res1, argh
152
153         pop argh pop argl
154         pop tmp                 ;
155         out SREG, tmp         ; Pop from stack
156         pop tmp                 ;
157         ret
158
159 write_2_nibbles:; {{{
160         push r24
161         ; save r24(LCD_Data)
162         in r25 ,PIND; read PIND
163         andi r25 ,0x0f
164         andi r24 ,0xf0
165         add r24 ,r25
166         out PORTD ,r24;
167         ; r24[3:0] Holds previus PORTD[3:0]
168         ; r24[7:4] <-- LCD_Data_High_Byte
169         ;
170         sbi PORTD ,PD3
171         nop
172         nop
173         cbi PORTD ,PD3; Enable Pulse
174         pop r24
175         swap r24
176         andi r24 ,0xf0
177         add r24 ,r25
178         out PORTD ,r24; Recover r24(LCD_Data)
179         ;
180         ; r24[3:0] Holds previus PORTD[3:0]
181         ; r24[7:4] <-- LCD_Data_Low_Byte
182         sbi PORTD ,PD3
```

```
        nop
        nop
        cbi PORTD ,PD3; Enable Pulse
        ret; }}}

lcd_data:; {{{
        sbi PORTD ,PD2
        rcall write_2_nibbles
        ldi r24 ,250
        ldi r25 ,0
        rcall wait_usec
        ret; }}}

lcd_command:; {{{
        cbi PORTD ,PD2
        rcall write_2_nibbles
        ldi r24 ,250
        ldi r25 ,0
        rcall wait_usec
        ret; }}}

lcd_clear_display:; {{{
        ldi r24 ,0x01
        rcall lcd_command        ; clear display command
        ldi r24 ,low(5)
        ldi r25 ,high(5)
        rcall wait_msec                ; Wait 5 mSec
        ret; }}}

lcd_init:; {{{
        ldi r24 ,low(200)
        ldi r25 ,high(200)
        rcall wait_msec;
        ; Wait 200 mSec
        ;
        ldi r24 ,0x30
        out PORTD ,r24
        sbi PORTD ,PD3
        nop
        nop
        cbi PORTD ,PD3
        ldi r24 ,250
        ldi r25 ,0
        rcall wait_usec; command to switch to 8 bit mode
        ;
        ; Enable Pulse
        ldi r24 ,0x30
        out PORTD ,r24
        sbi PORTD ,PD3
        nop
        nop
```

```
        cbi PORTD ,PD3
        ldi r24 ,250
        ldi r25 ,0
        rcall wait_usec; command to switch to 8 bit mode
        ;
        ; Enable Pulse
        ldi r24 ,0x30
        out PORTD ,r24
        sbi PORTD ,PD3
        nop
        nop
        cbi PORTD ,PD3
        ldi r24 ,250
        ldi r25 ,0
        rcall wait_usec

        ldi r24 ,0x20
        out PORTD ,r24
        sbi PORTD ,PD3
        nop
        nop
        cbi PORTD ,PD3
        ldi r24 ,250
        ldi r25 ,0
        rcall wait_usec; command to switch to 4 bit mode
        ldi r24 ,0x28
        rcall lcd_command; 5x8 dots, 2 lines
        ldi r24 ,0x0c
        rcall lcd_command
        rcall lcd_clear_display
        ldi r24 ,0x06
        rcall lcd_command
        ret; }}}

/* wait_msec:
        push r24 ; 2 cycles
        push r25 ; 2 cycles
        ldi r24 , low(999) ; 1 cycle
        ldi r25 , high(999) ; 1 cycle
        rcall wait_usec ; 998.375 usec
        pop r25 ; 2 cycles
        pop r24 ; 2 cycles
        nop ; 1 cycle
        nop ; 1 cycle
        sbiw r24 , 1 ; 2 cycles
        brne wait_msec ; 1 or 2 cycles
        ret ; 4 cycles */
wait_msec: ; delay of 1000*F1+6 cycles (almost equal to 1000*F1 cycles){{{
; total delay of next 4 instruction group = 1+(249*4-1) = 996 cycles
        ldi r23, 249                        ; (1 cycle)
loop_inn:
```

```
        dec r23                                         ; 1 cycle
        nop                                                 ; 1 cycle
        brne loop_inn                           ; 1 or 2 cycles

        sbiw r24, 1                                     ; 2 cycles
        brne wait_msec                          ; 1 or 2 cycles

        ret
wait_usec:
        sbiw r24 ,1 ; 2 cycles (2/16 usec)
        call delay_8cycles ; 4+8=12 cycles
        brne wait_usec ; 1 or 2 cycles
        ret
delay_8cycles:
        nop
        nop
        nop
        ret
```

2 Ζήτημα 4.2

Στην άσκηση αυτή, αρχικά, μετατρέπουμε τον δοσμένο κώδικα Assembly για τον χειρισμό της οθόνης σε C. Για να τυπώνουμε αριθμούς προσθέτουμε συναρτήσεις `lcd_digit` και `lcd_number` που τυπώνουν 1 δεκαδικό ψηφίο και 1 32 bit μη προσημασμένο δεκαδικό αριθμό αντίστοιχα. Στον κύριο κώδικα αρχικοποιήσουμε το `PORTD` ως έξοδο, την οθόνη, και τον ADC με $V_{REF} = 5V$, είσοδο `ADC1` , δεξιά στοιχισμένη, χωρίς διακοπές και με συχνότητα 125kHz. Στη συνέχεια, διαβάζουμε διαρκώς την έξοδο του ADC και μόλις τελειώνει η μετατροπή εμφανίζουμε κάθε φορά την τιμή $\frac{ADC}{1024}V_{REF}$ με ακρίβεια 2 δεκαδικών στην οθόνη.

```
/*
 * main.c
 *
 * Created: 10/31/2024 1:20:10 AM
 *  Author: User
 */

#include <xc.h>

#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define NOP() do { __asm__ __volatile__ ( "nop "); } while (0)
#define V_REF 5

void write_2_nibbles(uint8_t data){
        uint8_t temp = PIND & 0x0f;
        PORTD = data&(0xf0) | temp;

        PORTD |= (1<<PD3);
```

```
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);

        PORTD = (data<<4)&(0xf0) | temp;

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

void lcd_data(uint8_t data){
        PORTD |= (1<<PD2);
        write_2_nibbles(data);
        _delay_us(250);
}

void lcd_command(uint8_t instruction){
        PORTD &= ~(1<<PD2);
        write_2_nibbles(instruction);
        _delay_us(250);
}

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

void lcd_init(){
        _delay_ms(200);

        PORTD = 0x30;
        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);
        _delay_us(250);

        PORTD = 0x30;
        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);
        _delay_us(250);

        PORTD = 0x30;
        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);
```

```
        _delay_us(250);

        lcd_command(0x28);

        lcd_command(0x0c);

        lcd_clear_display();

        lcd_command(0x06);
}

void lcd_digit(uint8_t digit){
        lcd_data(0x30 + digit);
}

void lcd_number(uint32_t number){
        uint8_t digits[10];
        int i = 0;
        if(number == 0){
                lcd_digit(0);
                return;
        }
        do{
                digits[i++] = number%10;
                number /= 10;
        } while(number > 0);
        for(; i > 0; ) lcd_digit(digits[--i]);
}

int main(void)
{
        DDRD = 0xff; //set PORTD as output
        lcd_init();
        _delay_ms(100);

        //   Vref = 5V, ADC1, Right adjust
        ADMUX = (1 << REFS0) | (0 << ADLAR) | (1 << MUX0);
        //   Enable, no interrupt, no conversion, 125 kHz
        ADCSRA = (1 << ADEN) | (0 << ADSC) | (0 << ADIE) | (7 << ADPS0);

        uint32_t val;

    while(1)
    {
        lcd_clear_display();
                _delay_ms(1000);
                ADCSRA |= 1 << ADSC;
                while (ADCSRA & (1 << ADSC));
                val = (((uint32_t)ADC)*V_REF*100)>>10;
                lcd_number(val/100);
                lcd_data('.');
```

```
                lcd_number(val%100);
                _delay_ms(1000);
    }
}
```

3 Ζήτημα 4.3

Κρατάμε τον κώδικα C για τον χειρισμό της οθόνης και την αρχικοποίηση της εξόδου (αυτή τη φορά `PORTD` και `PORTB`), της οθόνης και του ADC (αυτή τη φορά με είσοδο το `ADC2`). Αρχικοποιούμε επιπλέον τον μετρητή `Timer1` σε λειτουργία CTC για να μετράει 100ms και να κάνει διακοπή κάθε φορά που αυτά περνάνε. Για να μην ξοδεύουμε πολύ χρόνο μέσα στη ρουτίνα εξυπηρέτησης της διακοπής, κατά τις διακοπές αλλάζουμε τιμή σε μια boolean σημαία που υποδεικνύει ότι έγινε διακοπή από τον μετρητή. Στον κύριο κώδικα, ελέγχουμε αν αυτή η σημαία είναι σηκωμένη, και, αν ναι, αφού τη χαμηλώσουμε, διαβάζουμε την τιμή του ADC και υπολογίζουμε την τάση $V_{gas} = \frac{ADC}{1024} V_{REF}$. Στη συνέχεια, υπολογίζουμε τα ppm μέσω της συνάρτησης `calc_ppm` και, ανάλογα με την τιμή του ψηφίου των εκατοντάδων των ppm ανάβουμε το αντίστοιχο LED του `PORTB` . Αν ανινχνεύονται πάνω από 70ppm εμφανίζουμε στην οθόνη το μήνυμα `"GAS DETECTED"` και αναβοσβήνουμε το αναμμένο LED του `PORTB` , ενώ αλλιώς εμφανίζουμε στην οθόνη `"CLEAR"` και το αντίστοιχο αναμμένο LED του `PORTB` παραμένει στάσιμο.

```
/*
 * main.c
 *
 * Created: 10/31/2024 11:29:07 PM
 *  Author: User
 */

#include <xc.h>

#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdbool.h>

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

void write_2_nibbles(uint8_t data){
        uint8_t temp = PIND & 0x0f;
        PORTD = data&(0xf0) | temp;

        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);

        PORTD = (data<<4)&(0xf0) | temp;

        PORTD |= (1<<PD3);
        NOP();
        NOP();
```

```
        PORTD &= ~(1<<PD3);
}

void lcd_data(uint8_t data){
        PORTD |= (1<<PD2);
        write_2_nibbles(data);
        _delay_us(250);
}

void lcd_command(uint8_t instruction){
        PORTD &= ~(1<<PD2);
        write_2_nibbles(instruction);
        _delay_us(250);
}

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

void lcd_init(){
        _delay_ms(200);

        PORTD = 0x30;
        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);
        _delay_us(250);

        PORTD = 0x30;
        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);
        _delay_us(250);

        PORTD = 0x30;
        PORTD |= (1<<PD3);
        NOP();
        NOP();
        PORTD &= ~(1<<PD3);
        _delay_us(250);

        lcd_command(0x28);

        lcd_command(0x0c);

        lcd_clear_display();

        lcd_command(0x06);
```

```
}

void lcd_digit(uint8_t digit){
        lcd_data(0x30 + digit);
}

void lcd_number(uint32_t number){
        uint8_t digits[10];
        int i = 0;
        if(number == 0){
                lcd_digit(0);
                return;
        }
        do{
                digits[i++] = number%10;
                number /= 10;
        } while(number > 0);
        for(; i > 0; ) lcd_digit(digits[--i]);
}

uint16_t calc_ppm(float V_gas){
        if(V_gas <= 0.1) return 0;
        uint16_t val = 10000.0/129.0*(V_gas - 0.1);
        if(val > 500) return 500;
        else return val;
}

bool interrupted = true;

ISR(TIMER1_COMPA_vect){
        interrupted = true;
        return;
}

void lcd_char(char c){
        lcd_data(c);
}

void lcd_string(char *s){
        while(*s != '\0'){
                lcd_char(*s);
                s++;
        }
}

int main(void)
{
        DDRD = 0xff; //set PORTD as output
        DDRB = 0xff; //set PORTB as output
        lcd_init();
        _delay_ms(100);
```

```
134
135         //   Vref = 5V, ADC2, Right adjust
136         ADMUX = (1 << REFS0) | (0 << ADLAR) | (1 << MUX1);
137         //   Enable, no interrupt, no conversion, 125 kHz
138         ADCSRA = (1 << ADEN) | (0 << ADSC) | (0 << ADIE) | (7 << ADPS0);
139
140         // Init Timer1
141         TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10);                 // CTC
    ↪ mode, pre-scaler 1024
142         OCR1A =
    ↪ 1562;
    ↪ Compare value for 100ms: 1562/16000000*1024 = 0.099968 = 100ms
143         TIMSK1 |= (1 <<
    ↪ OCIE1A);                                                               //
    ↪ Enable interrupt
144
145         uint8_t prev_PORTB = 0;
146         sei();
147
148         uint16_t ppm = 0;
149
150         while(1)
151         {
152                 // flicker
153                 _delay_ms(50);
154                 if (ppm > 70) PORTB = 0;
155                 _delay_ms(50);
156                 PORTB = prev_PORTB;
157
158                 if(!interrupted) continue;
159                 interrupted = false;
160
161                 // _delay_ms(1000);
162                 ADCSRA |= 1 << ADSC;
163                 while (ADCSRA & (1 << ADSC));
164                 float V_gas = ADC*5.0/(1<<10);
165                 ppm = calc_ppm(V_gas);
166                 lcd_clear_display();
167                 if(ppm <= 70){
168                         PORTB = 1;
169                         char str[6] = "CLEAR\0";
170                         lcd_string(str);
171                 } else {
172                         char str[13] = "GAS DETECTED\0";
173                         lcd_string(str);
174                         if(ppm <= 100) PORTB = 1;
175                         else if(ppm <= 200) PORTB = 2;
176                         else if(ppm <= 300) PORTB = 4;
177                         else if(ppm <= 400) PORTB = 8;
178                         else if(ppm <= 500) PORTB = 16;
179                         else PORTB = 32;
```

```
180
181                     }
182                     prev_PORTB = PORTB;
183                     // _delay_ms(1000);
184             }
185     }
```