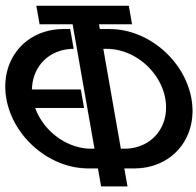


ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ
УНИВЕРЗИТЕТ У БАЊОЈ ЛУЦИ

Пројектовање дигиталних система

Студент: Марко Грумић
Број индекса: 1120/11

Предметни наставник: др Бранко Докић
Предметни асистент: Жељеко Ивановић

Број бодова:

Бања Лука
Септембар, 2016. године

РЕАЛИЗАЦИЈА XTEA АЛГОРИТМА

Марко Грумић
E-mail: marko.grumic@live.com

1. ДЕФИНИСАЊЕ ПРОЈЕКТА

У оквиру пројектног задатка потребно је остварити двосмјерну комуникацију између два уређаја користећи RS232 протокол за комуникацију. Додатно, податке које размјењују уређаји треба криптовати XTEA симетричним алгоритмом за енкрипцију. За потребе пројектног задатка уређаји су микроконтролер и персонални рачунар. На персоналном рачунару треба написати програм који ће комуницирати са микроконтролером путем RS232 протокола, док на страни микроконтолера треба испројектовати коло са 2x16 LCD дисплејем у 4-битном режиму на који ће се приказивати карактери послати са персоналног рачунара. Претпоставка је да је кључ за енкрипцију/декрипцију већ размијењен.

2. СПЕЦИФИКАЦИЈА

2.1. Спецификација софтвера

- За дизајнирање електричне шеме и симулацију кориштен је ISIS Proteus 7.6.
- За развој програма за микроконтролер кориштен је развојно окружење MIDE-51.
- MikroElektronika 8051 Flash Programmer за програмирање микроконтролера
- Програмски језик Ц, у оквиру развојног окружења Code::Blocks и MinGW gcc 4.9.3
- RS232 библиотека за комуникацију са RS232 COM портом (http://www.teuniz.net/RS-232/RS-232.tar.gz) под GNU GPL v3 лиценцом

2.2. Спецификација хардвера

За реализацију пројекта кориштено је:

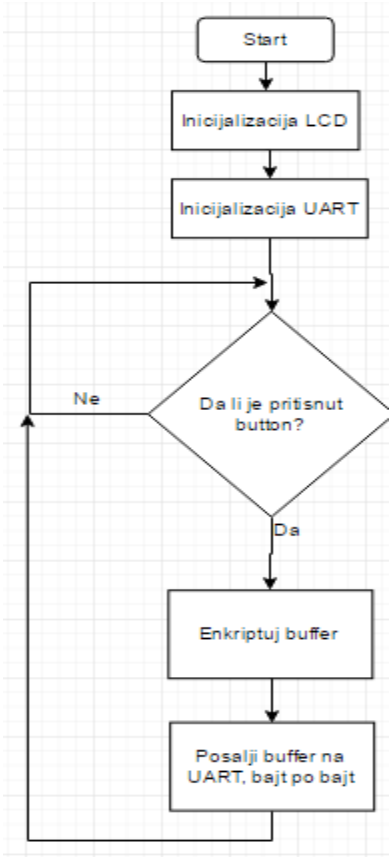
- Atmel AT89C52/AT89S8253 микроконтролер (симулација/тестирање)
- MikroElektronika Easy8051 v6 развојно окружење
- LM016L Smart LCD дисплеј 2 линије по 16 знакова

3. ОПИС ДИЗАЈНА

3.1. Увод

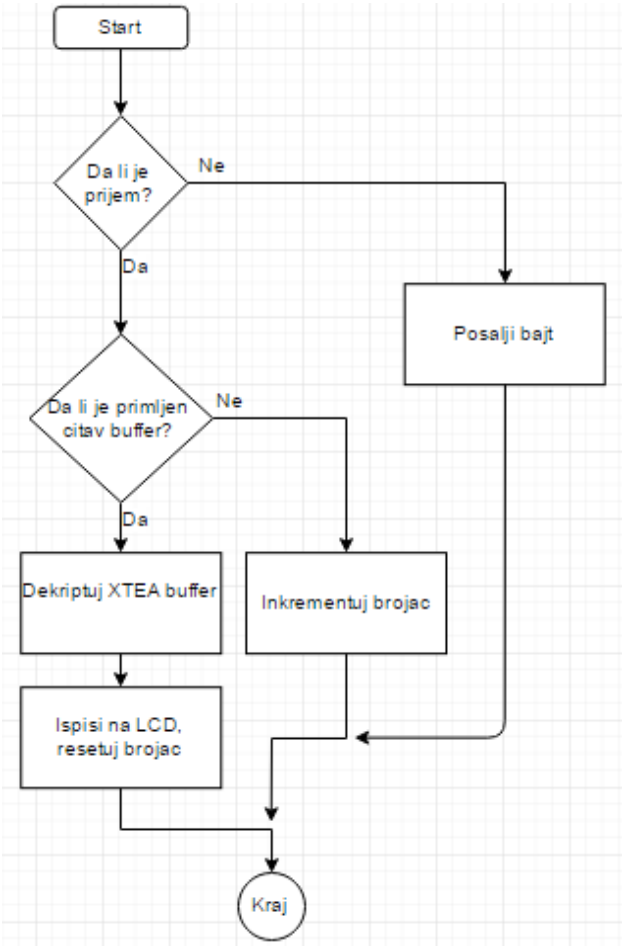
С' обзиром да је такт на развојном окрзжењу 10MHz baud rate са најмањим одступањем је 300bps. Програм на страни рачунара користи конзолу и за пријем и за слање података, до колизије не може доћи приликом исписа јер је пријем/слање окружено “mutexima”. Како XTEA алгоритам криптује увијек фиксну дужину података (8 бајтова) ради лакше имплементације криптује се само један бајт и шаље, док је осталих 7 бајтова 0.

3.2. Дијаграми тока извршавања



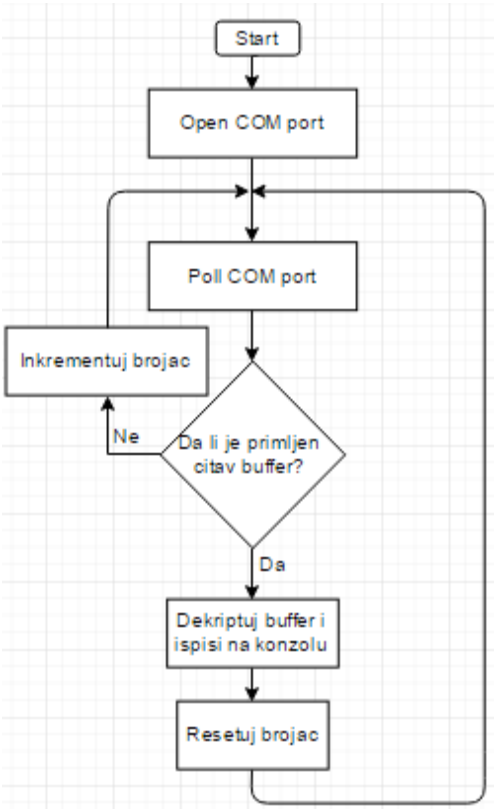
Сл.1. *Дијаграм тока основног програма микроконтролера*

На претходној слици приказан је дијаграм тока извршавања основног програма на микроконтролеру. Програм након иницијализације LCD-а и серијске комуникације улази у бесконачну петљу и ради “polling” предефинисаних дугмади који су накачени на одређене пинове микроконтролера. У случају да је неки од дугмади притиснут, одговарајући карактер се енкриптује и шаље на UART бајт по бајт.



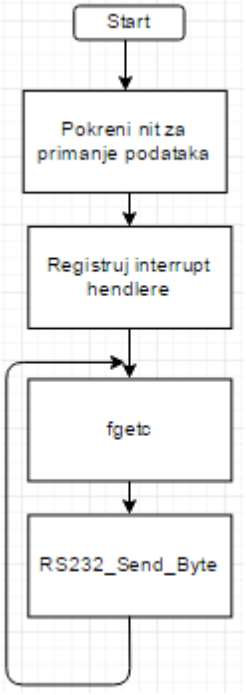
Сл.2. Дијаграм тока прекидне сервисне рутине серијске комуникације

На претходној слици је приказан дијаграм тока прекидне сервисне рутине серијске комуникације. У случају пријема, смјештамо бајт по бајт у локални бафер, а када попунимо XTEA бафер (када добијемо 8 бајтова) декриптујемо бафер и исписујемо добијени карактер на LCD.



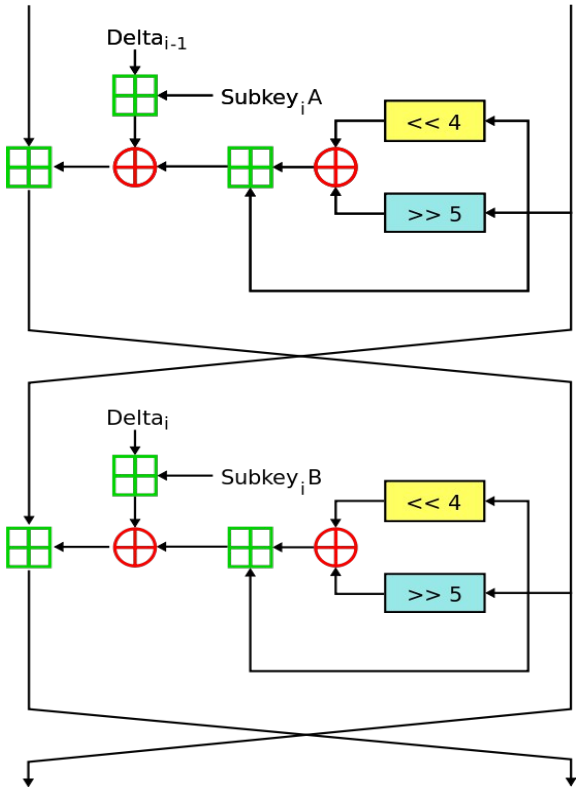
Сл.3. Дијаграм тока нити која прима податке

На претходној слици је приказан дијаграм тока дијела програма који прима податке на персоналном рачунару. У питању је нит која се покреће из основног програма и врши “polling” из отвореног тока према RS232 порту. Када прими 8 бајтова ради се декрипција бафера и исписује се декриптован карактер на стандардни излаз.



Сл.4. Дијаграм тока нити основног програма

На претходној слици ја приказан дијаграм тока основне нити програма на персоналном рачунару који покреће нит за пријем података са RS232, региструје "interrupt" хендлере и онда улази у бесконачну петљу гдје чита са стандардног улаза карактер по карактер и шаље их на RS232.



Сл. 5. Дијаграм тока једне рунде XTEA алгоритма

На слици 5 је приказана једна рунда алгоритма. Препоручен број рунди од стране аутора је 32 или 64. С' обзиром да је у питању симетручни крипто-алгоритам у локалној меморији чувамо кључ који размјењујемо са другом страном. Бафер који криптујемо има 64 бита, дијели се у два 32-битна података. Постоји вриједност делта која се користи да би додатно "замиксали" енкриипцију која је константа (0x9E3779B9). У листингу се налази имплементација XTEA алгоритма у C програмском језику и у асемблеру за архитектуру 8051.

4. ИМПЛЕМЕНТАЦИЈА

4.1. Увод

Први корак реализације пројекта био је упознавање са основним карактеристикама компоненти које се користе у оквиру пројекта. Након тога дизајнирање електричне шеме и програмског кода и на крају, симулација и тестирање. Пројекат је верификован и валидиран на развојном окрзжењу. Ове фазе су се понављале у итерацијама.

4.2. Реализација

Прво сам се упознао са LCD дисплејем у 4-битном режиму. Затим сам приступио дизајнирању електричн шеме. Пинове дисплеја сам повезао на пинове микроконтролера на порту 1 (како је писало у упутству за развојно окрзжење), као што се може видјети у прилогу. С' обзиром да на порту 1 имамо уграђене *pullup* отпорнике који ограничавају струју на ред mA није било потребе за додатним *pullup* отпорницима.

Затим сам написао продецуре RESET_LCD, INIT_DISPLAY, LCD_CMD и LCD_DATA. *RESET_LCD* ресетује LCD дисплеј у смислу да иницијализује дисплеј у 4-битном режиму. *INIT_DISPLAY* брише садржај дисплеја, поставља курсор на почетну позицију и укључује курсор. *LCD_CMD* шаље дисплеју команду на одговарајуће пинове, док *LCD_DATA* ради исту ствар, осим што се RS битом дисплеја регулише да ли је примљени бајт *data* или *command*. У случају да је примљени податак типа *data* исти се исписује на прву слободну позицију дисплеја. Након тога написао сам процедуру *ISPIS* која узима карактер из меморијске локације која се налази на адреси *rec_char* и исписује на прву слободну позицију на дисплеју. Када се дисплеј попуни, брише се и исписује се поново на почетну позицију. Након верификације рада са *LCD* дисплејем прешао сам на *RS232* комуникацију.

У софтверу за симулацију постоји компонента за COM порт на основу које сам тестирао комуникацију микроконтролера са *RS232* кроз виртуелне портове и клијентске софтвере за

читање/писање на COM порт. Са обзиром да је фреквенција микроконтролера на развојном окружењу 10MHz било је потребно конфигурисати тајмер 1 да ради као окидач за серијску комуникацију. Baudrate са најмањом девијацијом за фреквенцију 10MHz је 300bps. Тајмер 1 је подешен да ради у аутоматском моду који генерише прекиде сходно вриједностима фреквенције и одабраног baudrate-a. Након тога било је потребно написати прекидну сервисну рутину за серијску комуникацију. За почетак прекидна сервисна рутина је само слала податке на порт и примала и исписивала на дисплеј карактер који прими (што је била верификација да серијска комуникација ради). У оквиру серијске комуникације ту је рутина за сервисирање прекида серијске комуникације *SERIJSKA* и процедура за слање бафера на серијски порт *SEND_BYTES*.

Кад смо имали све компоненте које су нам потребне (дисплеј, серијска комуникација) долази до имплементације крипто-алгоритма (у нашем случају XTEA – eXtended Tiny Encryption Algorithm). Kako је XTEA алгоритам који ради са 8-бајтним бафером, ради лакше имплементације, изабрано је да се криптује само један карактер при једном слању, што значи да је први бајт карактер који се шаље, док су остали 0. На пријемној страни се тај бафер декриптује и исписује се на конзолу на персоналном рачунару. Претходно је претпостављено да је кључ размијењен тако да је на пријемној и предајној страни кључ смјештен у локалној меморији. Пошто требамо примати/слати 8 бајтова по карактеру било је потребно измијенити прекидну рутину серијске комуникације. На пријему смо додали бројач који након сваких 8 примљених бајтова декриптује бафер и узима најнижи бајт и исписује га на дисплеј. На предајној страни након обраде притиснутог карактера (криптовање и припремање бафера за слање) слали 8 бајтова у бафер за сериску комуникацију, бајт по бајт. У сврху енкрипције/декрипције постоје процедуре EXTEA и DXTEA које криптују/декриптују бафер који се налази на адреси “y0-y4” и “z1-z4” док се кључ налази на адреси “Key”.

На страни рачунара било је потребно написати програм који ће примати/слати енкриптоване карактере. Са обзиром да нам је на располагању конзолна апликација и да нам је потребно "истовремено" слање и примање бафера дизајн програма је морао бити вишенитни. Основна нит је дизајнирана тако да на почетку покреће нит која је задужена за пријем података, након тога улази у бесконачну петљу и прозива COM порт да ли је стигло 8 бајтова (тзв. полинг). Када стигне 8 бајтова, декриптује се бафер и исписује на конзолу. Истовремено друга нит ослушкује стандардни улаз и када се нађе нешто на баферу стандардног улаза та нит узима карактер по карактер и шаље га на COM порт. Не постоји могућност да дође до колизије код пријема и предаје зато што су нити заштићене "*mutexima"* (mutex – mutual exclusive, узајамно искључиви).

4.3. Резултати симулације

Што се тиче симулације у Proteus-у је успјешно прошла комуникација између микроконтролера и клијента за серијску комуникацију преко виртуелних портова, након тога упоредно је писана процедура за енкрипцију са истим кључем на рачунару и на микроконтролеру гдје сам успио добити исте вриједности криптованог карактера (ту је било доста посла да се утврди који бајт треба да иде на коју позицију и који бајт из кључа је бајт највише вриједности). Када је одрађена енкрипција, искористио сам горе поменуту библиотеку како бих повезао програм на рачунару и на микроконтролеру кроз RS232 виртуелни порт како бих добио двосмјерну комуникацију, што је на крају крајева и био смисао пројектног задатка. Помоћу осцилоскопа у Proteus-у успио сам да измјерим вријеме енкрипције и декрипције на микроконтролеру које износи око 20ms при фреквенцији од 10MHz.

5. РЕВИЗИЈЕ И КОМЕНТАРИ

При изради пројектног задатка консултовао сам се са асистентом Жељком Ивановићем и колегом Милошем Ковачевићем.

Листинг 1. Програмски код микроконтолера

```
;
*****************************************************************************************
;*
;*
;*        SERIJSKA KOMUNIKACIJA SA PC RACUNAROM PREKO RS232 sa XTEA enkripcijom
;*
;*
;
*****************************************************************************************
E       EQU     p2.1                    ; definisanje simbolickog imena za pin p2.1
RS      EQU     p2.0                    ; definisanje simbolickog imena za pin p2.0
                                        ; RW nepotreban, radimo samo upisivanje
D4      EQU     p2.2                    ; DATA pinovi LCD display-a
D5      EQU     p2.3
D6      EQU     p2.4
D7      EQU     p2.5
lcd_port EQU p2                                 ; definisanje simbolickog imena za lcd port

Button_A EQU p1.0                       ; predefinisan pin za slovo 'A'
Button_W EQU p1.1                       ; predefinisan pin za slovo 'W'
Button_S EQU p1.2                       ; predefinisan pin za slovo 'S'
Button_D EQU p1.3                       ; predefinisan pin za slovo 'D'
;-----------------------------------------------------------------------------------------------------------------
;                                               DATA SEGMENT
;-----------------------------------------------------------------------------------------------------------------
DSEG  AT  30h
y0:   DS    1   ;y0 - y3 prva cetiri bajta XTEA buffer-a
y1:   DS    1
y2:   DS    1
y3:   DS    1
z0:   DS    1   ;z0 - z3 druga cetiri bajta XTEA buffer-a
z1:   DS    1
z2:   DS    1
z3:   DS    1
```

```
tmp0: DS    1   ;tmp buffer za potrebe XTEA enkripcije/dekripcije
tmp1: DS    1
tmp2: DS    1
tmp3: DS    1
sum0: DS    1   ;sum - za mixanje sum vrijednosti i za shuffelovanje
sum1: DS    1   ;indeksa bajta kljuca
sum2: DS    1
sum3: DS    1
temp: DS    1   ;tmp pomocna promjenjiva
disp_counter: DS    1 ; counter modula 16 koliko karaktera smo do sada upisali na LCD
rec_char:     DS    1 ; promjenjiva u koju smjestamo primljeni dekriptovan karakter
rec_byte:     DS    1 ; counter modula 8 koliko bajtova smo primili preko RS232
;-------------------------------------------------------------------------------------------------------------------
;                                                   KODNI SEGMENT
;-------------------------------------------------------------------------------------------------------------------

CSEG

        org 0000h
                jmp POCETAK
        org 0003h
                reti
        org 000bh
                reti
        org 0013h
                reti
        org 001bh
                reti
        org 0023h
                jmp SERIJSKA ; serijska prekidna servisna rutina
        org 002bh
                reti
```

```
ORG 0050h


POCETAK:
        call init_display   ; pozivamo potprogram za inicijalizaciju displeja
        mov rec_byte, #00h
        mov p0, #0FFh
        mov disp_counter, #80h
        mov p1, #0FFh
        mov     tmod,#20h       ; tajmer 1 u auto modu
        mov     th1,#0A9h       ; 300 baud rate
        mov     scon,#50h       ; inicijalizacija serijske komunikacije
        mov     ie,#90h         ; omogucenje prekida serijske komunikacije
        setb    tr1             ; startovanje tajmera 1
        mov r3, #00h
        jmp PETLJA


; Potprogram za resetovanje XTEA buffera
NULIRAJ:
        mov y0, #00h
        mov y1, #00h
        mov y2, #00h
        mov y3, #00h
        mov z0, #00h
        mov z1, #00h
        mov z2, #00h
        mov z3, #00h
        ret


; Potprogram za slanje XTEA buffera na RS232
; delay nakon slanja svakog bajta, inace ne stigne prekidna rutina da ga obradi
; i posalje na RS232
SEND_BYTES:
        mov sbuf, y0
```

```
        call kasnjenje
        mov sbuf, y1
        call kasnjenje
        mov sbuf, y2
        call kasnjenje
        mov sbuf, y3
        call kasnjenje
        mov sbuf, z0
        call kasnjenje
        mov sbuf, z1
        call kasnjenje
        mov sbuf, z2
        call kasnjenje
        mov sbuf, z3
        call kasnjenje
        ret
;-------------------------------------------------------------------------------------------------------
;
;                   PROGRAM SE VRTI U OVOJ PETLJI
;
;-------------------------------------------------------------------------------------------------------
PETLJA:
        mov r7, p1
        cjne r7, #0FFh, GO_GO_GO ; provjeravamo da li je stisnut bilo koji karakter za slanje
        mov r3, #00h
        jmp PETLJA
GO_GO_GO:
        call NULIRAJ

        cjne r3, #00h, PETLJA
        mov r3, #01h

        mov a, p1 ; provjeravamo koji karakter je pritisnut
```

```
        anl a, #001h
        jz A_PUSHED

        mov a, p1
        anl a, #002h
        jz W_PUSHED

        mov a, p1
        anl a, #004h
        jz S_PUSHED

        mov a, p1
        anl a, #008h
        jz D_PUSHED

        jmp PETLJA
A_PUSHED:
        mov y0, #041h
        jmp END_THIS
W_PUSHED:
        mov y0, #057h
        jmp END_THIS
S_PUSHED:
        mov y0, #053h
        jmp END_THIS
D_PUSHED:
        mov y0, #044h
        ;jmp END_THIS ; u sustini nam ne treba jmp ovde
END_THIS:
        call EXTea
        call SEND_BYTES
        jmp PETLJA
```

```
;-------------------------------------------------------------
;
;                POTPROGRAM ZA OPSLUZIVANJE PREKIDA SERIJSKOG PORTA
;
;-------------------------------------------------------------


SERIJSKA:
        jb ri, PRIJEM     ; ako je ri setovan radi se o prijemu u suprotnom o predaji podatka
        clr ti            ; mora se softverski obrisati
        RETI              ; vracanje iz potprograma


PRIJEM:
        mov rec_char, sbuf ; preuzimamo karakter iz sbuf
        mov a, rec_byte
        cjne a, #00h, dalje1 ; provjeravamo koji karakter smo dobili
        mov y0, rec_char
        jmp die_ende
dalje1:
        cjne a, #01h, dalje2
        mov y1, rec_char
        jmp die_ende
dalje2:
        cjne a, #02h, dalje3
        mov y2, rec_char
        jmp die_ende
dalje3:
        cjne a, #03h, dalje4
        mov y3, rec_char
        jmp die_ende
dalje4:
        cjne a, #04h, dalje5
        mov z0, rec_char
        jmp die_ende
```

```
dalje5:
        cjne a, #05h, dalje6
        mov z1, rec_char
        jmp die_ende
dalje6:
        cjne a, #06h, dalje7
        mov z2, rec_char
        jmp die_ende
dalje7:
        mov z3, rec_char
        call DXtea
        mov rec_char, y0
        call ispis
        mov rec_byte, #00h
        clr ri
        RETI
die_ende:
        inc rec_byte ; povecavamo broj primljenih bajtova
        clr ri              ; mora se softverski obrisati
        RETI


RESET_LCD:
        mov lcd_port, #11111111b
        call kasnjenje

        mov lcd_port, #00001110b
        call kasnjenje

        mov lcd_port, #00001100b
        call kasnjenje

        mov lcd_port, #00001110b
        call kasnjenje
```

```
        mov lcd_port, #00001100b
        call kasnjenje

        mov lcd_port, #00001110b
        call kasnjenje

        mov lcd_port, #00001100b
        call kasnjenje

        mov lcd_port, #00001010b
        call kasnjenje

        mov lcd_port, #00001000b
        call kasnjenje

        ret

LCD_CMD:
        mov temp, a
        swap a
        anl a, #0Fh
        rl a
        rl a
        add a, #02h
        mov lcd_port, a
        clr E
        call kasnjenje

        mov a, temp
        anl a, #0Fh
        rl a
        rl a
        add a, #02h
```

```
        mov lcd_port, a
        clr E
        call kasnjenje
ret

LCD_DATA:
        mov temp, a
        swap a
        anl a, #0Fh
        rl a
        rl a
        add a, #03h
        mov lcd_port, a
        nop
        clr E
        call kasnjenje

        mov a, temp
        anl a, #0Fh
        rl a
        rl a
        add a, #03h
        mov lcd_port, a
        nop
        clr E
        call kasnjenje
ret

INIT_DISPLAY:
        call RESET_LCD
        mov a, #028h
        call LCD_CMD
        mov a, #01h
```

```
        call LCD_CMD
        mov a, #0Fh
        call LCD_CMD
        mov a, #006h
        call LCD_CMD
        mov a, #080h
        call LCD_CMD
 RET

;=========================================================================
;
;                   POTPROGRAM KOJI ISPISUJE KARAKTER IZ AKUMULATORA NA LCD-a
;
;=========================================================================

ISPIS:
        mov r0, disp_counter
        cjne r0, #090h, nope1 ; kraj prvog reda
        mov r0, #0C0h ;
        jmp nope
        nope1:
        cjne r0, #0D0h, nope ; kraj drugog reda
        mov a, #01h ; brisemo display jer je popunjen
        call LCD_CMD
        mov r0, #080h
        nope:
        mov     a,r0
        call LCD_CMD
        mov     a, rec_char
        call LCD_DATA
        inc r0
        mov disp_counter, r0
RET
```

```
;-------------------------------------------------------------------------------------------------------
;
;                       XTEA encrypt
;
;-------------------------------------------------------------------------------------------------------


EXTea:
     clr   a
     mov   sum0,a   ;sum = 0
     mov   sum1,a
     mov   sum2,a
     mov   sum3,a
      mov   r2,#32*2   ;nr of rounds *2 (because of trick with twice the main code, one for y and one for z; and another inside...)


     mov   dptr,#key  ;dptr se ne mijenja
ETeaRound:


     mov   r4,z0
     mov   r5,z1
     mov   r6,z2
     mov   r7,z3


ETeaSubRound:
     mov   r0,#tmp3    ;tmp = z << 4
     mov   a,r7
     swap  a
     mov   @r0,a           ;@r0=tmp3
     mov   a,r6
     swap  a
     xchd  a,@r0           ;@r0=tmp3
     dec   r0
```

```
mov   @r0,a          ;@r0=tmp2
mov   a,r5
swap  a
xchd  a,@r0          ;@r0=tmp2
dec   r0
mov   @r0,a          ;@r0=tmp1
mov   a,r4
swap  a
xchd  a,@r0          ;@r0=tmp1
mov   tmp0,a
anl   tmp0,#0F0h

rrc   a              ;tmp ^=  z >> 5
anl   a,#07h
xrl   a,tmp3
xch   a,tmp3
rrc   a
xrl   a,tmp2
xch   a,tmp2
rrc   a
xrl   a,@r0  ;tmp1
xch   a,@r0  ;tmp1
rrc   a
xrl   a,tmp0



add   a,r4        ;z = z+tmp
mov   r4,a
mov   a,r5
addc  a,tmp1
mov   r5,a
mov   a,r6
addc  a,tmp2
```

```
    mov   r6,a
    mov   a,r7
    addc  a,tmp3
    mov   r7,a

    mov   a,r2
    jb    acc.0,ETeaX1
    mov   a,sum0        ;r0 = [sum&3]
    rl    a
    rl    a
    sjmp  ETeaX2
ETeaX1:
    mov   a,sum1        ;r0 = [sum>>11&3]
    rr    a
ETeaX2:
    anl   a,#0Ch
    mov   r0,a

    movc  a,@a+dptr     ;result ^= sum + k[pointer]
    inc   r0
    add   a,sum0
    xrl   a,r4
    mov   r4,a
    mov   a,r0
    movc  a,@a+dptr
    inc   r0
    addc  a,sum1
    xrl   a,r5
    mov   r5,a
    mov   a,r0
    movc  a,@a+dptr
    inc   r0
    addc  a,sum2
```

```
    xrl   a,r6
    mov   r6,a
    mov   a,r0
    movc  a,@a+dptr
    addc  a,sum3
    xrl   a,r7
    mov   r7,a

    dec   r2
    mov   a,r2
    jb    acc.0,ETeaSubRound2

    mov   a,r4
    add   a,z0
    mov   z0,a
    mov   a,r5
    addc  a,z1
    mov   z1,a
    mov   a,r6
    addc  a,z2
    mov   z2,a
    mov   a,r7
    addc  a,z3
    mov   z3,a

    cjne  r2,#0,ETeaRoundA
    ret
ETeaRoundA:
    jmp   ETeaRound

ETeaSubRound2:
    mov   a,r4
    add   a,y0
```

```
    mov   y0,a
    mov   r4,a
    mov   a,r5
    addc  a,y1
    mov   y1,a
    mov   r5,a
    mov   a,r6
    addc  a,y2
    mov   y2,a
    mov   r6,a
    mov   a,r7
    addc  a,y3
    mov   y3,a
    mov   r7,a

    mov   a,sum0   ;sum += delta
    add   a,#0B9h    ;delta[0]
    mov   sum0,a
    mov   a,sum1
    addc  a,#079h    ;delta[1]
    mov   sum1,a
    mov   a,sum2
    addc  a,#037h    ;delta[2]
    mov   sum2,a
    mov   a,sum3
    addc  a,#09Eh    ;delta[3]
    mov   sum3,a

    jmp   ETeaSubRound
;-------------------------------------------------------------------------------------------------------------
;
;                   XTEA decrypt
;
```

```
;-------------------------------------------------------------------------------------------------------------------

DXTea:

     mov   r2,#32*2   ;nr of rounds *2 (because of trick with twice the main code, one for y and one for z; and another inside...)

     mov   sum3,#0C6h

     mov   sum2,#0EFh

     mov   sum1,#037h

     mov   sum0,#020h


     mov   dptr,#key  ;dptr se ne mijanja

DTeaRound:


     mov   r4,y0

     mov   r5,y1

     mov   r6,y2

     mov   r7,y3


DTeaSubRound:

     mov   r0,#tmp3    ;tmp = y << 4

     mov   a,r7

     swap  a

     mov   @r0,a           ;@r0=tmp3

     mov   a,r6

     swap  a

     xchd  a,@r0           ;@r0=tmp3

     dec   r0

     mov   @r0,a           ;@r0=tmp2

     mov   a,r5

     swap  a

     xchd  a,@r0           ;@r0=tmp2

     dec   r0

     mov   @r0,a           ;@r0=tmp1

     mov   a,r4

     swap  a
```

```
xchd  a,@r0          ;@r0=tmp1
mov   tmp0,a
anl   tmp0,#0F0h

rrc   a              ;tmp ^=  y >> 5
anl   a,#07h
xrl   a,tmp3
xch   a,tmp3
rrc   a
xrl   a,tmp2
xch   a,tmp2
rrc   a
xrl   a,@r0  ;tmp1
xch   a,@r0  ;tmp1
rrc   a
xrl   a,tmp0

add   a,r4         ;y = y+tmp
mov   r4,a
mov   a,r5
addc  a,tmp1
mov   r5,a
mov   a,r6
addc  a,tmp2
mov   r6,a
mov   a,r7
addc  a,tmp3
mov   r7,a

mov   a,r2
jnb   acc.0,DTeaX1
mov   a,sum0         ;r0 = [sum&3]
rl    a
```

```
    rl    a
    sjmp  DTeaX2
DTeaX1:
    mov   a,sum1        ;r0 = [sum>>11&3]
    rr    a
DTeaX2:
    anl   a,#0Ch
    mov   r0,a

    movc  a,@a+dptr     ;result ^= sum + k[pointer]
    inc   r0
    add   a,sum0
    xrl   a,r4
    mov   r4,a
    mov   a,r0
    movc  a,@a+dptr
    inc   r0
    addc  a,sum1
    xrl   a,r5
    mov   r5,a
    mov   a,r0
    movc  a,@a+dptr
    inc   r0
    addc  a,sum2
    xrl   a,r6
    mov   r6,a
    mov   a,r0
    movc  a,@a+dptr
    addc  a,sum3
    xrl   a,r7
    mov   r7,a

    dec   r2
```

```
    mov   a,r2
    jb    acc.0,DTeaSubRound2

    clr   c
    mov   a,y0
    subb  a,r4
    mov   y0,a
    mov   a,y1
    subb  a,r5
    mov   y1,a
    mov   a,y2
    subb  a,r6
    mov   y2,a
    mov   a,y3
    subb  a,r7
    mov   y3,a

    cjne  r2,#0,DTeaRoundA
    ret
DTeaRoundA:
    jmp   DTeaRound

DTeaSubRound2:
    clr   c
    mov   a,z0
    subb  a,r4
    mov   z0,a
    mov   r4,a
    mov   a,z1
    subb  a,r5
    mov   z1,a
    mov   r5,a
    mov   a,z2
```

```
    subb  a,r6
    mov   z2,a
    mov   r6,a
    mov   a,z3
    subb  a,r7
    mov   z3,a
    mov   r7,a

    clr   c
    mov   a,sum0   ;sum += delta
    subb  a,#0B9h    ;delta[0]
    mov   sum0,a
    mov   a,sum1
    subb  a,#079h    ;delta[1]
    mov   sum1,a
    mov   a,sum2
    subb  a,#037h    ;delta[2]
    mov   sum2,a
    mov   a,sum3
    subb  a,#09Eh    ;delta[3]
    mov   sum3,a

    jmp   DTeaSubRound




;-------------------------------------------------------------------------------------------------------------
;
;                       End XTEA
;
;-------------------------------------------------------------------------------------------------------------

Key:
```

```
    db    09fh, 012h, 0abh, 099h
    db    0fah, 0e6h, 0e1h, 04dh
    db    000h, 0b1h, 0e8h, 0bbh
    db    0f3h, 08eh, 088h, 0fah
;KEY: 0x9f12ab99
;    0xfae6e14d
;    0x00b1e8bb
;    0xf38e88fa


;=========================================================================
;
;         POTPROGRAM ZA KASNJENJE
;
;=========================================================================
KASNJENJE:
         mov     r4      ,       #1
KASNJENJE3:
         mov     r2      ,       #200
KASNJENJE2:
         mov     r1      ,       #210
KASNJENJE1:
         djnz    r1      ,       KASNJENJE1
         djnz    r2      ,       KASNJENJE2
         djnz    r4      ,       KASNJENJE3
         ret


END
```

Листинг 2. Програмски код програма на персоналном рачунару

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <pthread.h>

#include "rs232.h"

#define NUM_ROUNDS 32
#define MAX_STR 4096

pthread_mutex_t* mymutex = NULL;
//#define _XTEA_DEBUG
typedef enum _COM_PORT {
   COM1 = 0,
   COM2,
   COM3,
   COM4,
   COM5,
   COM6,
   COM7,
   COM8,
   COM9,
   COM10,
   COM11,
   COM12,
   COM13,
   COM14,
   COM15,
   COM16
} COM_PORT;

uint32_t key[4] = {
```

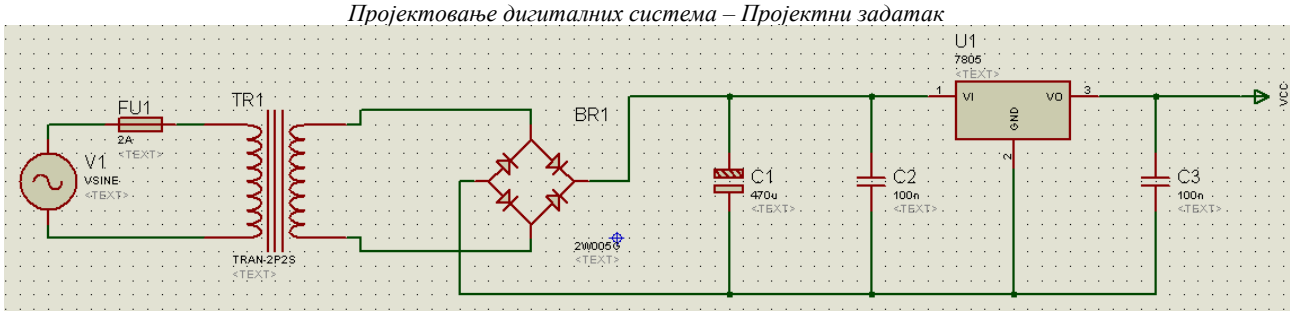
```
    0x99ab129f,
    0x4de1e6fa,
    0xbbe8b100,
    0xfa888ef3
};

static const COM_PORT CURRENT_COM_PORT = COM1;
static const uint32_t BAUD_RATE = 300;

/* 64 bita podataka u v[0] - v[1] i 128 bita kljuca u key[0] - key[3] */
void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
    }
    v[0]=v0; v[1]=v1;
}

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0]=v0; v[1]=v1;
}

void *listening( void *ptr ) {
```

```
uint32_t dataRec[2] = {0x00000000, 0x00000000};
uint8_t recBytes = 0;
uint8_t buf[4096] = {0};

int err = RS232_OpenComport(CURRENT_COM_PORT, BAUD_RATE, "8N1");
if (err == 1) {
   RS232_CloseComport(CURRENT_COM_PORT);
   return (void*)0;
}
while(1) {
   int n, i;
   n = RS232_PollComport(CURRENT_COM_PORT, buf, 4096);
   if (recBytes == 0 && n) {
      pthread_mutex_lock(mymutex);
   }
   #ifdef _XTEA_DEBUG
   for (i = 0; i < n && recBytes < 8; i++) {
      fprintf(stderr, "Received byte: '0x%02X'\n", buf[i]);
   }
   #else
   for (i = 0; i < n && recBytes < 8; i++) {
      ((uint8_t*)dataRec)[recBytes++] = buf[i];
   }
   if(recBytes == 8) {
   #ifdef _XTEA_DEBUG
      fprintf(stderr, "Received block: 0x%08X 0x%08X\n", dataRec[0], dataRec[1]);
   #endif
      decipher(NUM_ROUNDS, dataRec, key);
      fprintf(stderr, "Received character: '%c'\n", ((uint8_t*)dataRec)[0]);
      recBytes = 0;
      pthread_mutex_unlock(mymutex);
   }
   #endif // _XTEA_DEBUG
```

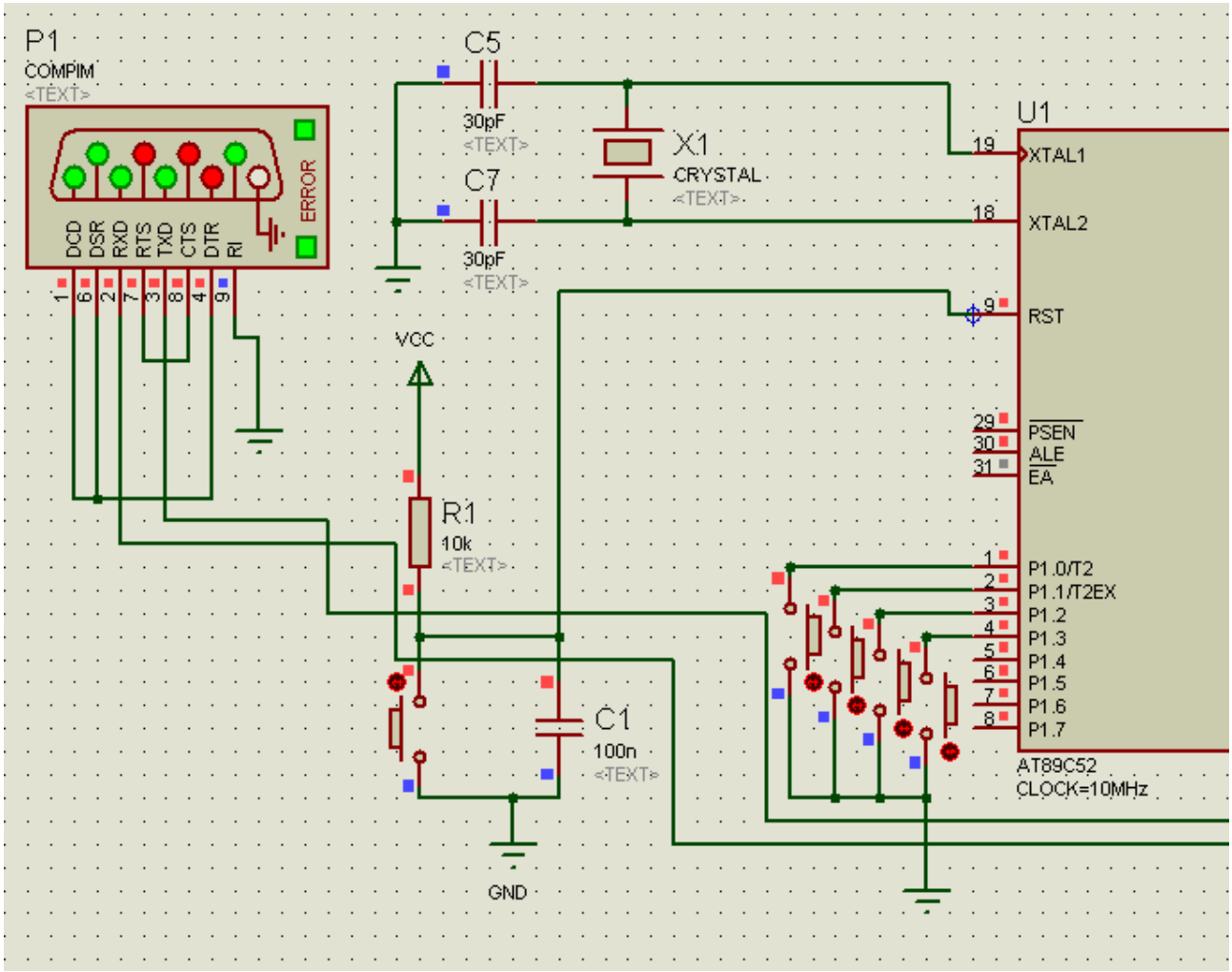
```
      Sleep(100);
    }
}
BOOL CtrlHandler( DWORD fdwCtrlType )
{
  switch( fdwCtrlType )
  {
    // Handle the CTRL-C signal.
    case CTRL_C_EVENT:
      printf( "Exiting...\n\n");
      Beep( 750, 300 );
      RS232_CloseComport(CURRENT_COM_PORT);
      pthread_mutex_destroy(mymutex);
      free(mymutex);
      exit(0);
      return( TRUE );

    // CTRL-CLOSE: confirm that the user wants to exit.
    case CTRL_CLOSE_EVENT:
      Beep( 600, 200 );
      return( TRUE );

    // Pass other signals to the next handler.
    case CTRL_BREAK_EVENT:
      Beep( 900, 200 );
      return FALSE;

    case CTRL_LOGOFF_EVENT:
      Beep( 1000, 200 );
      return FALSE;

    case CTRL_SHUTDOWN_EVENT:
      Beep( 750, 500 );
```

```
    return FALSE;

  default:
    return FALSE;
 }
}
int main() {
  pthread_t listening_thread;
  int iret1;
  uint32_t buffer[2] = {0};
  mymutex = (pthread_mutex_t*) malloc (sizeof(pthread_mutex_t));
  pthread_mutex_init(mymutex, NULL);
  iret1 = pthread_create( &listening_thread, NULL, listening, (void*)"");
  if(iret1) {
    fprintf(stderr,"Error - pthread_create() return code: %d\n",iret1);
    exit(EXIT_FAILURE);
  } else {
    fprintf(stderr,"Listening thread created... Success\n");
  }

  if(SetConsoleCtrlHandler((PHANDLER_ROUTINE)CtrlHandler, TRUE)) {
    printf( "\nThe Control Handler is installed... Success\n" );
  }

  while(1) {
    char c = fgetc(stdin);
    pthread_mutex_lock(mymutex);
    int i;
    if (c > 0x21 || c == ' ') {
      buffer[0] = (uint32_t) c;
      encipher(NUM_ROUNDS, buffer, key);
      for(i = 0; i < 8; i++) {
        RS232_SendByte(CURRENT_COM_PORT, ((uint8_t*)buffer)[i]);
```

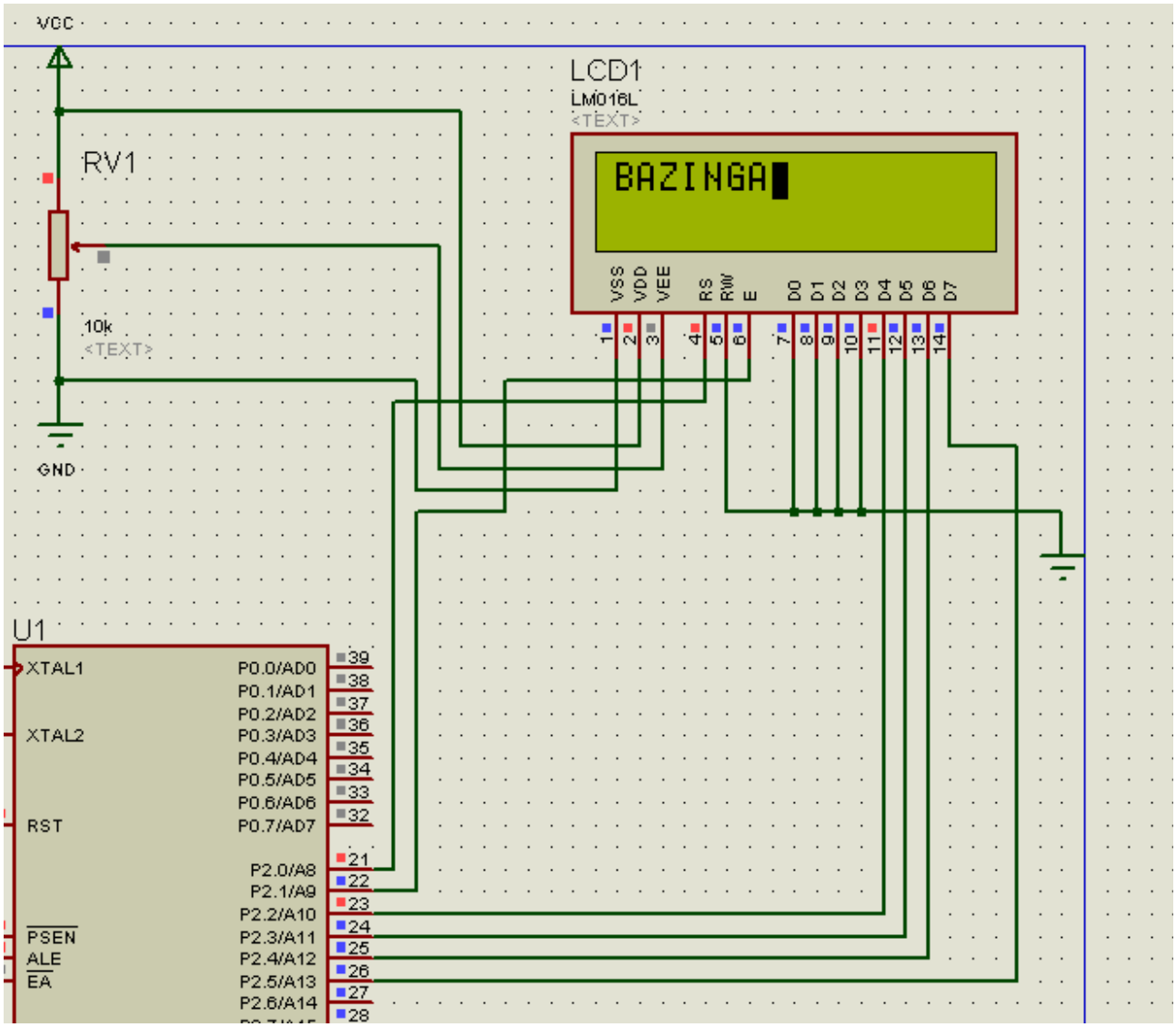
```
            Sleep(100);
        }
        fprintf(stderr, "Sent char: %c (0x%08X%08X)\n", c, buffer[0], buffer[1]);
        buffer[0] = 0;
        buffer[1] = 0;
    }
    pthread_mutex_unlock(mymutex);
    Sleep(1000);
  }
}
```



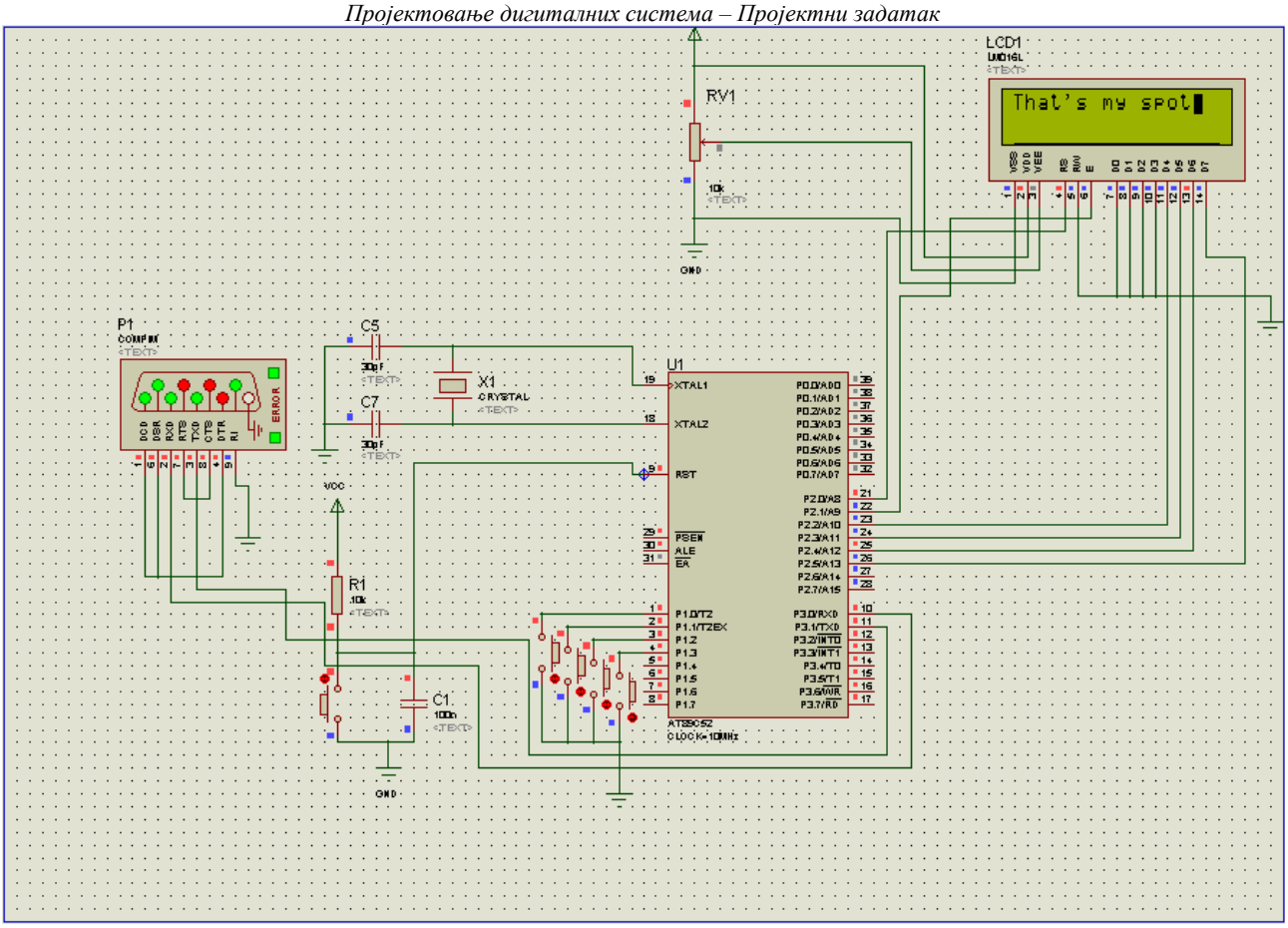
Сл. 6. Електрична шема напајања



Сл. 7. Коло за ресет, осцилатор и RS232 комуникацију



Сл. 8. Повезивање LCD дисплеја са микроконтролером



Сл. 9. Шема комплетног пројекта